C-------------------------------------------------------------------

C

C HERE BEGINS THE DVODE PACKAGE

C

C-------------------------------------------------------------------

C

\*DECK DVODE

SUBROUTINE DVODE (F, NEQ, Y, T, TOUT, ITOL, RTOL, ATOL, ITASK,

1 ISTATE, IOPT, RWORK, LRW, IWORK, LIW, JAC, MF,

2 RPAR, IPAR)

EXTERNAL F, JAC

DOUBLE PRECISION Y, T, TOUT, RTOL, ATOL, RWORK, RPAR

INTEGER NEQ, ITOL, ITASK, ISTATE, IOPT, LRW, IWORK, LIW,

1 MF, IPAR

DIMENSION Y(\*), RTOL(\*), ATOL(\*), RWORK(LRW), IWORK(LIW),

1 RPAR(\*), IPAR(\*)

C-----------------------------------------------------------------------

C DVODE.. Variable-coefficient Ordinary Differential Equation solver,

C with fixed-leading-coefficient implementation.

C This version is in double precision.

C

C DVODE solves the initial value problem for stiff or nonstiff

C systems of first order ODEs,

C dy/dt = f(t,y) , or, in component form,

C dy(i)/dt = f(i) = f(i,t,y(1),y(2),...,y(NEQ)) (i = 1,...,NEQ).

C DVODE is a package based on the EPISODE and EPISODEB packages, and

C on the ODEPACK user interface standard, with minor modifications.

C-----------------------------------------------------------------------

C Revision History (YYMMDD)

C 890615 Date Written

C 890922 Added interrupt/restart ability, minor changes throughout.

C 910228 Minor revisions in line format, prologue, etc.

C 920227 Modifications by D. Pang:

C (1) Applied subgennam to get generic intrinsic names.

C (2) Changed intrinsic names to generic in comments.

C (3) Added \*DECK lines before each routine.

C 920721 Names of routines and labeled Common blocks changed, so as

C to be unique in combined single/double precision code (ACH).

C 920722 Minor revisions to prologue (ACH).

C 920831 Conversion to double precision done (ACH).

C 921106 Fixed minor bug: ETAQ,ETAQM1 in DVSTEP SAVE statement (ACH).

C 921118 Changed LUNSAV/MFLGSV to IXSAV (ACH).

C 941222 Removed MF overwrite; attached sign to H in estimated second

C derivative in DVHIN; misc. comment corrections throughout.

C 970515 Minor corrections to comments in prologue, DVJAC.

C-----------------------------------------------------------------------

C References..

C

C 1. P. N. Brown, G. D. Byrne, and A. C. Hindmarsh, "VODE: A Variable

C Coefficient ODE Solver," SIAM J. Sci. Stat. Comput., 10 (1989),

C pp. 1038-1051. Also, LLNL Report UCRL-98412, June 1988.

C 2. G. D. Byrne and A. C. Hindmarsh, "A Polyalgorithm for the

C Numerical Solution of Ordinary Differential Equations,"

C ACM Trans. Math. Software, 1 (1975), pp. 71-96.

C 3. A. C. Hindmarsh and G. D. Byrne, "EPISODE: An Effective Package

C for the Integration of Systems of Ordinary Differential

C Equations," LLNL Report UCID-30112, Rev. 1, April 1977.

C 4. G. D. Byrne and A. C. Hindmarsh, "EPISODEB: An Experimental

C Package for the Integration of Systems of Ordinary Differential

C Equations with Banded Jacobians," LLNL Report UCID-30132, April

C 1976.

C 5. A. C. Hindmarsh, "ODEPACK, a Systematized Collection of ODE

C Solvers," in Scientific Computing, R. S. Stepleman et al., eds.,

C North-Holland, Amsterdam, 1983, pp. 55-64.

C 6. K. R. Jackson and R. Sacks-Davis, "An Alternative Implementation

C of Variable Step-Size Multistep Formulas for Stiff ODEs," ACM

C Trans. Math. Software, 6 (1980), pp. 295-318.

C-----------------------------------------------------------------------

C Authors..

C

C Peter N. Brown and Alan C. Hindmarsh

C Center for Applied Scientific Computing, L-561

C Lawrence Livermore National Laboratory

C Livermore, CA 94551

C and

C George D. Byrne

C Illinois Institute of Technology

C Chicago, IL 60616

C-----------------------------------------------------------------------

C Summary of usage.

C

C Communication between the user and the DVODE package, for normal

C situations, is summarized here. This summary describes only a subset

C of the full set of options available. See the full description for

C details, including optional communication, nonstandard options,

C and instructions for special situations. See also the example

C problem (with program and output) following this summary.

C

C A. First provide a subroutine of the form..

C

C SUBROUTINE F (NEQ, T, Y, YDOT, RPAR, IPAR)

C DOUBLE PRECISION T, Y, YDOT, RPAR

C DIMENSION Y(NEQ), YDOT(NEQ)

C

C which supplies the vector function f by loading YDOT(i) with f(i).

C

C B. Next determine (or guess) whether or not the problem is stiff.

C Stiffness occurs when the Jacobian matrix df/dy has an eigenvalue

C whose real part is negative and large in magnitude, compared to the

C reciprocal of the t span of interest. If the problem is nonstiff,

C use a method flag MF = 10. If it is stiff, there are four standard

C choices for MF (21, 22, 24, 25), and DVODE requires the Jacobian

C matrix in some form. In these cases (MF .gt. 0), DVODE will use a

C saved copy of the Jacobian matrix. If this is undesirable because of

C storage limitations, set MF to the corresponding negative value

C (-21, -22, -24, -25). (See full description of MF below.)

C The Jacobian matrix is regarded either as full (MF = 21 or 22),

C or banded (MF = 24 or 25). In the banded case, DVODE requires two

C half-bandwidth parameters ML and MU. These are, respectively, the

C widths of the lower and upper parts of the band, excluding the main

C diagonal. Thus the band consists of the locations (i,j) with

C i-ML .le. j .le. i+MU, and the full bandwidth is ML+MU+1.

C

C C. If the problem is stiff, you are encouraged to supply the Jacobian

C directly (MF = 21 or 24), but if this is not feasible, DVODE will

C compute it internally by difference quotients (MF = 22 or 25).

C If you are supplying the Jacobian, provide a subroutine of the form..

C

C SUBROUTINE JAC (NEQ, T, Y, ML, MU, PD, NROWPD, RPAR, IPAR)

C DOUBLE PRECISION T, Y, PD, RPAR

C DIMENSION Y(NEQ), PD(NROWPD,NEQ)

C

C which supplies df/dy by loading PD as follows..

C For a full Jacobian (MF = 21), load PD(i,j) with df(i)/dy(j),

C the partial derivative of f(i) with respect to y(j). (Ignore the

C ML and MU arguments in this case.)

C For a banded Jacobian (MF = 24), load PD(i-j+MU+1,j) with

C df(i)/dy(j), i.e. load the diagonal lines of df/dy into the rows of

C PD from the top down.

C In either case, only nonzero elements need be loaded.

C

C D. Write a main program which calls subroutine DVODE once for

C each point at which answers are desired. This should also provide

C for possible use of logical unit 6 for output of error messages

C by DVODE. On the first call to DVODE, supply arguments as follows..

C F = Name of subroutine for right-hand side vector f.

C This name must be declared external in calling program.

C NEQ = Number of first order ODE-s.

C Y = Array of initial values, of length NEQ.

C T = The initial value of the independent variable.

C TOUT = First point where output is desired (.ne. T).

C ITOL = 1 or 2 according as ATOL (below) is a scalar or array.

C RTOL = Relative tolerance parameter (scalar).

C ATOL = Absolute tolerance parameter (scalar or array).

C The estimated local error in Y(i) will be controlled so as

C to be roughly less (in magnitude) than

C EWT(i) = RTOL\*abs(Y(i)) + ATOL if ITOL = 1, or

C EWT(i) = RTOL\*abs(Y(i)) + ATOL(i) if ITOL = 2.

C Thus the local error test passes if, in each component,

C either the absolute error is less than ATOL (or ATOL(i)),

C or the relative error is less than RTOL.

C Use RTOL = 0.0 for pure absolute error control, and

C use ATOL = 0.0 (or ATOL(i) = 0.0) for pure relative error

C control. Caution.. Actual (global) errors may exceed these

C local tolerances, so choose them conservatively.

C ITASK = 1 for normal computation of output values of Y at t = TOUT.

C ISTATE = Integer flag (input and output). Set ISTATE = 1.

C IOPT = 0 to indicate no optional input used.

C RWORK = Real work array of length at least..

C 20 + 16\*NEQ for MF = 10,

C 22 + 9\*NEQ + 2\*NEQ\*\*2 for MF = 21 or 22,

C 22 + 11\*NEQ + (3\*ML + 2\*MU)\*NEQ for MF = 24 or 25.

C LRW = Declared length of RWORK (in user's DIMENSION statement).

C IWORK = Integer work array of length at least..

C 30 for MF = 10,

C 30 + NEQ for MF = 21, 22, 24, or 25.

C If MF = 24 or 25, input in IWORK(1),IWORK(2) the lower

C and upper half-bandwidths ML,MU.

C LIW = Declared length of IWORK (in user's DIMENSION statement).

C JAC = Name of subroutine for Jacobian matrix (MF = 21 or 24).

C If used, this name must be declared external in calling

C program. If not used, pass a dummy name.

C MF = Method flag. Standard values are..

C 10 for nonstiff (Adams) method, no Jacobian used.

C 21 for stiff (BDF) method, user-supplied full Jacobian.

C 22 for stiff method, internally generated full Jacobian.

C 24 for stiff method, user-supplied banded Jacobian.

C 25 for stiff method, internally generated banded Jacobian.

C RPAR,IPAR = user-defined real and integer arrays passed to F and JAC.

C Note that the main program must declare arrays Y, RWORK, IWORK,

C and possibly ATOL, RPAR, and IPAR.

C

C E. The output from the first call (or any call) is..

C Y = Array of computed values of y(t) vector.

C T = Corresponding value of independent variable (normally TOUT).

C ISTATE = 2 if DVODE was successful, negative otherwise.

C -1 means excess work done on this call. (Perhaps wrong MF.)

C -2 means excess accuracy requested. (Tolerances too small.)

C -3 means illegal input detected. (See printed message.)

C -4 means repeated error test failures. (Check all input.)

C -5 means repeated convergence failures. (Perhaps bad

C Jacobian supplied or wrong choice of MF or tolerances.)

C -6 means error weight became zero during problem. (Solution

C component i vanished, and ATOL or ATOL(i) = 0.)

C

C F. To continue the integration after a successful return, simply

C reset TOUT and call DVODE again. No other parameters need be reset.

C

C-----------------------------------------------------------------------

C EXAMPLE PROBLEM

C

C The following is a simple example problem, with the coding

C needed for its solution by DVODE. The problem is from chemical

C kinetics, and consists of the following three rate equations..

C dy1/dt = -.04\*y1 + 1.e4\*y2\*y3

C dy2/dt = .04\*y1 - 1.e4\*y2\*y3 - 3.e7\*y2\*\*2

C dy3/dt = 3.e7\*y2\*\*2

C on the interval from t = 0.0 to t = 4.e10, with initial conditions

C y1 = 1.0, y2 = y3 = 0. The problem is stiff.

C

C The following coding solves this problem with DVODE, using MF = 21

C and printing results at t = .4, 4., ..., 4.e10. It uses

C ITOL = 2 and ATOL much smaller for y2 than y1 or y3 because

C y2 has much smaller values.

C At the end of the run, statistical quantities of interest are

C printed. (See optional output in the full description below.)

C To generate Fortran source code, replace C in column 1 with a blank

C in the coding below.

C

C EXTERNAL FEX, JEX

C DOUBLE PRECISION ATOL, RPAR, RTOL, RWORK, T, TOUT, Y

C DIMENSION Y(3), ATOL(3), RWORK(67), IWORK(33)

C NEQ = 3

C Y(1) = 1.0D0

C Y(2) = 0.0D0

C Y(3) = 0.0D0

C T = 0.0D0

C TOUT = 0.4D0

C ITOL = 2

C RTOL = 1.D-4

C ATOL(1) = 1.D-8

C ATOL(2) = 1.D-14

C ATOL(3) = 1.D-6

C ITASK = 1

C ISTATE = 1

C IOPT = 0

C LRW = 67

C LIW = 33

C MF = 21

C DO 40 IOUT = 1,12

C CALL DVODE(FEX,NEQ,Y,T,TOUT,ITOL,RTOL,ATOL,ITASK,ISTATE,

C 1 IOPT,RWORK,LRW,IWORK,LIW,JEX,MF,RPAR,IPAR)

C WRITE(6,20)T,Y(1),Y(2),Y(3)

C 20 FORMAT(' At t =',D12.4,' y =',3D14.6)

C IF (ISTATE .LT. 0) GO TO 80

C 40 TOUT = TOUT\*10.

C WRITE(6,60) IWORK(11),IWORK(12),IWORK(13),IWORK(19),

C 1 IWORK(20),IWORK(21),IWORK(22)

C 60 FORMAT(/' No. steps =',I4,' No. f-s =',I4,

C 1 ' No. J-s =',I4,' No. LU-s =',I4/

C 2 ' No. nonlinear iterations =',I4/

C 3 ' No. nonlinear convergence failures =',I4/

C 4 ' No. error test failures =',I4/)

C STOP

C 80 WRITE(6,90)ISTATE

C 90 FORMAT(///' Error halt.. ISTATE =',I3)

C STOP

C END

C

C SUBROUTINE FEX (NEQ, T, Y, YDOT, RPAR, IPAR)

C DOUBLE PRECISION RPAR, T, Y, YDOT

C DIMENSION Y(NEQ), YDOT(NEQ)

C YDOT(1) = -.04D0\*Y(1) + 1.D4\*Y(2)\*Y(3)

C YDOT(3) = 3.D7\*Y(2)\*Y(2)

C YDOT(2) = -YDOT(1) - YDOT(3)

C RETURN

C END

C

C SUBROUTINE JEX (NEQ, T, Y, ML, MU, PD, NRPD, RPAR, IPAR)

C DOUBLE PRECISION PD, RPAR, T, Y

C DIMENSION Y(NEQ), PD(NRPD,NEQ)

C PD(1,1) = -.04D0

C PD(1,2) = 1.D4\*Y(3)

C PD(1,3) = 1.D4\*Y(2)

C PD(2,1) = .04D0

C PD(2,3) = -PD(1,3)

C PD(3,2) = 6.D7\*Y(2)

C PD(2,2) = -PD(1,2) - PD(3,2)

C RETURN

C END

C

C The following output was obtained from the above program on a

C Cray-1 computer with the CFT compiler.

C

C At t = 4.0000e-01 y = 9.851680e-01 3.386314e-05 1.479817e-02

C At t = 4.0000e+00 y = 9.055255e-01 2.240539e-05 9.445214e-02

C At t = 4.0000e+01 y = 7.158108e-01 9.184883e-06 2.841800e-01

C At t = 4.0000e+02 y = 4.505032e-01 3.222940e-06 5.494936e-01

C At t = 4.0000e+03 y = 1.832053e-01 8.942690e-07 8.167938e-01

C At t = 4.0000e+04 y = 3.898560e-02 1.621875e-07 9.610142e-01

C At t = 4.0000e+05 y = 4.935882e-03 1.984013e-08 9.950641e-01

C At t = 4.0000e+06 y = 5.166183e-04 2.067528e-09 9.994834e-01

C At t = 4.0000e+07 y = 5.201214e-05 2.080593e-10 9.999480e-01

C At t = 4.0000e+08 y = 5.213149e-06 2.085271e-11 9.999948e-01

C At t = 4.0000e+09 y = 5.183495e-07 2.073399e-12 9.999995e-01

C At t = 4.0000e+10 y = 5.450996e-08 2.180399e-13 9.999999e-01

C

C No. steps = 595 No. f-s = 832 No. J-s = 13 No. LU-s = 112

C No. nonlinear iterations = 831

C No. nonlinear convergence failures = 0

C No. error test failures = 22

C-----------------------------------------------------------------------

C Full description of user interface to DVODE.

C

C The user interface to DVODE consists of the following parts.

C

C i. The call sequence to subroutine DVODE, which is a driver

C routine for the solver. This includes descriptions of both

C the call sequence arguments and of user-supplied routines.

C Following these descriptions is

C \* a description of optional input available through the

C call sequence,

C \* a description of optional output (in the work arrays), and

C \* instructions for interrupting and restarting a solution.

C

C ii. Descriptions of other routines in the DVODE package that may be

C (optionally) called by the user. These provide the ability to

C alter error message handling, save and restore the internal

C COMMON, and obtain specified derivatives of the solution y(t).

C

C iii. Descriptions of COMMON blocks to be declared in overlay

C or similar environments.

C

C iv. Description of two routines in the DVODE package, either of

C which the user may replace with his own version, if desired.

C these relate to the measurement of errors.

C

C-----------------------------------------------------------------------

C Part i. Call Sequence.

C

C The call sequence parameters used for input only are

C F, NEQ, TOUT, ITOL, RTOL, ATOL, ITASK, IOPT, LRW, LIW, JAC, MF,

C and those used for both input and output are

C Y, T, ISTATE.

C The work arrays RWORK and IWORK are also used for conditional and

C optional input and optional output. (The term output here refers

C to the return from subroutine DVODE to the user's calling program.)

C

C The legality of input parameters will be thoroughly checked on the

C initial call for the problem, but not checked thereafter unless a

C change in input parameters is flagged by ISTATE = 3 in the input.

C

C The descriptions of the call arguments are as follows.

C

C F = The name of the user-supplied subroutine defining the

C ODE system. The system must be put in the first-order

C form dy/dt = f(t,y), where f is a vector-valued function

C of the scalar t and the vector y. Subroutine F is to

C compute the function f. It is to have the form

C SUBROUTINE F (NEQ, T, Y, YDOT, RPAR, IPAR)

C DOUBLE PRECISION T, Y, YDOT, RPAR

C DIMENSION Y(NEQ), YDOT(NEQ)

C where NEQ, T, and Y are input, and the array YDOT = f(t,y)

C is output. Y and YDOT are arrays of length NEQ.

C (In the DIMENSION statement above, NEQ can be replaced by

C \* to make Y and YDOT assumed size arrays.)

C Subroutine F should not alter Y(1),...,Y(NEQ).

C F must be declared EXTERNAL in the calling program.

C

C Subroutine F may access user-defined real and integer

C work arrays RPAR and IPAR, which are to be dimensioned

C in the main program.

C

C If quantities computed in the F routine are needed

C externally to DVODE, an extra call to F should be made

C for this purpose, for consistent and accurate results.

C If only the derivative dy/dt is needed, use DVINDY instead.

C

C NEQ = The size of the ODE system (number of first order

C ordinary differential equations). Used only for input.

C NEQ may not be increased during the problem, but

C can be decreased (with ISTATE = 3 in the input).

C

C Y = A real array for the vector of dependent variables, of

C length NEQ or more. Used for both input and output on the

C first call (ISTATE = 1), and only for output on other calls.

C On the first call, Y must contain the vector of initial

C values. In the output, Y contains the computed solution

C evaluated at T. If desired, the Y array may be used

C for other purposes between calls to the solver.

C

C This array is passed as the Y argument in all calls to

C F and JAC.

C

C T = The independent variable. In the input, T is used only on

C the first call, as the initial point of the integration.

C In the output, after each call, T is the value at which a

C computed solution Y is evaluated (usually the same as TOUT).

C On an error return, T is the farthest point reached.

C

C TOUT = The next value of t at which a computed solution is desired.

C Used only for input.

C

C When starting the problem (ISTATE = 1), TOUT may be equal

C to T for one call, then should .ne. T for the next call.

C For the initial T, an input value of TOUT .ne. T is used

C in order to determine the direction of the integration

C (i.e. the algebraic sign of the step sizes) and the rough

C scale of the problem. Integration in either direction

C (forward or backward in t) is permitted.

C

C If ITASK = 2 or 5 (one-step modes), TOUT is ignored after

C the first call (i.e. the first call with TOUT .ne. T).

C Otherwise, TOUT is required on every call.

C

C If ITASK = 1, 3, or 4, the values of TOUT need not be

C monotone, but a value of TOUT which backs up is limited

C to the current internal t interval, whose endpoints are

C TCUR - HU and TCUR. (See optional output, below, for

C TCUR and HU.)

C

C ITOL = An indicator for the type of error control. See

C description below under ATOL. Used only for input.

C

C RTOL = A relative error tolerance parameter, either a scalar or

C an array of length NEQ. See description below under ATOL.

C Input only.

C

C ATOL = An absolute error tolerance parameter, either a scalar or

C an array of length NEQ. Input only.

C

C The input parameters ITOL, RTOL, and ATOL determine

C the error control performed by the solver. The solver will

C control the vector e = (e(i)) of estimated local errors

C in Y, according to an inequality of the form

C rms-norm of ( e(i)/EWT(i) ) .le. 1,

C where EWT(i) = RTOL(i)\*abs(Y(i)) + ATOL(i),

C and the rms-norm (root-mean-square norm) here is

C rms-norm(v) = sqrt(sum v(i)\*\*2 / NEQ). Here EWT = (EWT(i))

C is a vector of weights which must always be positive, and

C the values of RTOL and ATOL should all be non-negative.

C The following table gives the types (scalar/array) of

C RTOL and ATOL, and the corresponding form of EWT(i).

C

C ITOL RTOL ATOL EWT(i)

C 1 scalar scalar RTOL\*ABS(Y(i)) + ATOL

C 2 scalar array RTOL\*ABS(Y(i)) + ATOL(i)

C 3 array scalar RTOL(i)\*ABS(Y(i)) + ATOL

C 4 array array RTOL(i)\*ABS(Y(i)) + ATOL(i)

C

C When either of these parameters is a scalar, it need not

C be dimensioned in the user's calling program.

C

C If none of the above choices (with ITOL, RTOL, and ATOL

C fixed throughout the problem) is suitable, more general

C error controls can be obtained by substituting

C user-supplied routines for the setting of EWT and/or for

C the norm calculation. See Part iv below.

C

C If global errors are to be estimated by making a repeated

C run on the same problem with smaller tolerances, then all

C components of RTOL and ATOL (i.e. of EWT) should be scaled

C down uniformly.

C

C ITASK = An index specifying the task to be performed.

C Input only. ITASK has the following values and meanings.

C 1 means normal computation of output values of y(t) at

C t = TOUT (by overshooting and interpolating).

C 2 means take one step only and return.

C 3 means stop at the first internal mesh point at or

C beyond t = TOUT and return.

C 4 means normal computation of output values of y(t) at

C t = TOUT but without overshooting t = TCRIT.

C TCRIT must be input as RWORK(1). TCRIT may be equal to

C or beyond TOUT, but not behind it in the direction of

C integration. This option is useful if the problem

C has a singularity at or beyond t = TCRIT.

C 5 means take one step, without passing TCRIT, and return.

C TCRIT must be input as RWORK(1).

C

C Note.. If ITASK = 4 or 5 and the solver reaches TCRIT

C (within roundoff), it will return T = TCRIT (exactly) to

C indicate this (unless ITASK = 4 and TOUT comes before TCRIT,

C in which case answers at T = TOUT are returned first).

C

C ISTATE = an index used for input and output to specify the

C the state of the calculation.

C

C In the input, the values of ISTATE are as follows.

C 1 means this is the first call for the problem

C (initializations will be done). See note below.

C 2 means this is not the first call, and the calculation

C is to continue normally, with no change in any input

C parameters except possibly TOUT and ITASK.

C (If ITOL, RTOL, and/or ATOL are changed between calls

C with ISTATE = 2, the new values will be used but not

C tested for legality.)

C 3 means this is not the first call, and the

C calculation is to continue normally, but with

C a change in input parameters other than

C TOUT and ITASK. Changes are allowed in

C NEQ, ITOL, RTOL, ATOL, IOPT, LRW, LIW, MF, ML, MU,

C and any of the optional input except H0.

C (See IWORK description for ML and MU.)

C Note.. A preliminary call with TOUT = T is not counted

C as a first call here, as no initialization or checking of

C input is done. (Such a call is sometimes useful to include

C the initial conditions in the output.)

C Thus the first call for which TOUT .ne. T requires

C ISTATE = 1 in the input.

C

C In the output, ISTATE has the following values and meanings.

C 1 means nothing was done, as TOUT was equal to T with

C ISTATE = 1 in the input.

C 2 means the integration was performed successfully.

C -1 means an excessive amount of work (more than MXSTEP

C steps) was done on this call, before completing the

C requested task, but the integration was otherwise

C successful as far as T. (MXSTEP is an optional input

C and is normally 500.) To continue, the user may

C simply reset ISTATE to a value .gt. 1 and call again.

C (The excess work step counter will be reset to 0.)

C In addition, the user may increase MXSTEP to avoid

C this error return. (See optional input below.)

C -2 means too much accuracy was requested for the precision

C of the machine being used. This was detected before

C completing the requested task, but the integration

C was successful as far as T. To continue, the tolerance

C parameters must be reset, and ISTATE must be set

C to 3. The optional output TOLSF may be used for this

C purpose. (Note.. If this condition is detected before

C taking any steps, then an illegal input return

C (ISTATE = -3) occurs instead.)

C -3 means illegal input was detected, before taking any

C integration steps. See written message for details.

C Note.. If the solver detects an infinite loop of calls

C to the solver with illegal input, it will cause

C the run to stop.

C -4 means there were repeated error test failures on

C one attempted step, before completing the requested

C task, but the integration was successful as far as T.

C The problem may have a singularity, or the input

C may be inappropriate.

C -5 means there were repeated convergence test failures on

C one attempted step, before completing the requested

C task, but the integration was successful as far as T.

C This may be caused by an inaccurate Jacobian matrix,

C if one is being used.

C -6 means EWT(i) became zero for some i during the

C integration. Pure relative error control (ATOL(i)=0.0)

C was requested on a variable which has now vanished.

C The integration was successful as far as T.

C

C Note.. Since the normal output value of ISTATE is 2,

C it does not need to be reset for normal continuation.

C Also, since a negative input value of ISTATE will be

C regarded as illegal, a negative output value requires the

C user to change it, and possibly other input, before

C calling the solver again.

C

C IOPT = An integer flag to specify whether or not any optional

C input is being used on this call. Input only.

C The optional input is listed separately below.

C IOPT = 0 means no optional input is being used.

C Default values will be used in all cases.

C IOPT = 1 means optional input is being used.

C

C RWORK = A real working array (double precision).

C The length of RWORK must be at least

C 20 + NYH\*(MAXORD + 1) + 3\*NEQ + LWM where

C NYH = the initial value of NEQ,

C MAXORD = 12 (if METH = 1) or 5 (if METH = 2) (unless a

C smaller value is given as an optional input),

C LWM = length of work space for matrix-related data..

C LWM = 0 if MITER = 0,

C LWM = 2\*NEQ\*\*2 + 2 if MITER = 1 or 2, and MF.gt.0,

C LWM = NEQ\*\*2 + 2 if MITER = 1 or 2, and MF.lt.0,

C LWM = NEQ + 2 if MITER = 3,

C LWM = (3\*ML+2\*MU+2)\*NEQ + 2 if MITER = 4 or 5, and MF.gt.0,

C LWM = (2\*ML+MU+1)\*NEQ + 2 if MITER = 4 or 5, and MF.lt.0.

C (See the MF description for METH and MITER.)

C Thus if MAXORD has its default value and NEQ is constant,

C this length is..

C 20 + 16\*NEQ for MF = 10,

C 22 + 16\*NEQ + 2\*NEQ\*\*2 for MF = 11 or 12,

C 22 + 16\*NEQ + NEQ\*\*2 for MF = -11 or -12,

C 22 + 17\*NEQ for MF = 13,

C 22 + 18\*NEQ + (3\*ML+2\*MU)\*NEQ for MF = 14 or 15,

C 22 + 17\*NEQ + (2\*ML+MU)\*NEQ for MF = -14 or -15,

C 20 + 9\*NEQ for MF = 20,

C 22 + 9\*NEQ + 2\*NEQ\*\*2 for MF = 21 or 22,

C 22 + 9\*NEQ + NEQ\*\*2 for MF = -21 or -22,

C 22 + 10\*NEQ for MF = 23,

C 22 + 11\*NEQ + (3\*ML+2\*MU)\*NEQ for MF = 24 or 25.

C 22 + 10\*NEQ + (2\*ML+MU)\*NEQ for MF = -24 or -25.

C The first 20 words of RWORK are reserved for conditional

C and optional input and optional output.

C

C The following word in RWORK is a conditional input..

C RWORK(1) = TCRIT = critical value of t which the solver

C is not to overshoot. Required if ITASK is

C 4 or 5, and ignored otherwise. (See ITASK.)

C

C LRW = The length of the array RWORK, as declared by the user.

C (This will be checked by the solver.)

C

C IWORK = An integer work array. The length of IWORK must be at least

C 30 if MITER = 0 or 3 (MF = 10, 13, 20, 23), or

C 30 + NEQ otherwise (abs(MF) = 11,12,14,15,21,22,24,25).

C The first 30 words of IWORK are reserved for conditional and

C optional input and optional output.

C

C The following 2 words in IWORK are conditional input..

C IWORK(1) = ML These are the lower and upper

C IWORK(2) = MU half-bandwidths, respectively, of the

C banded Jacobian, excluding the main diagonal.

C The band is defined by the matrix locations

C (i,j) with i-ML .le. j .le. i+MU. ML and MU

C must satisfy 0 .le. ML,MU .le. NEQ-1.

C These are required if MITER is 4 or 5, and

C ignored otherwise. ML and MU may in fact be

C the band parameters for a matrix to which

C df/dy is only approximately equal.

C

C LIW = the length of the array IWORK, as declared by the user.

C (This will be checked by the solver.)

C

C Note.. The work arrays must not be altered between calls to DVODE

C for the same problem, except possibly for the conditional and

C optional input, and except for the last 3\*NEQ words of RWORK.

C The latter space is used for internal scratch space, and so is

C available for use by the user outside DVODE between calls, if

C desired (but not for use by F or JAC).

C

C JAC = The name of the user-supplied routine (MITER = 1 or 4) to

C compute the Jacobian matrix, df/dy, as a function of

C the scalar t and the vector y. It is to have the form

C SUBROUTINE JAC (NEQ, T, Y, ML, MU, PD, NROWPD,

C RPAR, IPAR)

C DOUBLE PRECISION T, Y, PD, RPAR

C DIMENSION Y(NEQ), PD(NROWPD, NEQ)

C where NEQ, T, Y, ML, MU, and NROWPD are input and the array

C PD is to be loaded with partial derivatives (elements of the

C Jacobian matrix) in the output. PD must be given a first

C dimension of NROWPD. T and Y have the same meaning as in

C Subroutine F. (In the DIMENSION statement above, NEQ can

C be replaced by \* to make Y and PD assumed size arrays.)

C In the full matrix case (MITER = 1), ML and MU are

C ignored, and the Jacobian is to be loaded into PD in

C columnwise manner, with df(i)/dy(j) loaded into PD(i,j).

C In the band matrix case (MITER = 4), the elements

C within the band are to be loaded into PD in columnwise

C manner, with diagonal lines of df/dy loaded into the rows

C of PD. Thus df(i)/dy(j) is to be loaded into PD(i-j+MU+1,j).

C ML and MU are the half-bandwidth parameters. (See IWORK).

C The locations in PD in the two triangular areas which

C correspond to nonexistent matrix elements can be ignored

C or loaded arbitrarily, as they are overwritten by DVODE.

C JAC need not provide df/dy exactly. A crude

C approximation (possibly with a smaller bandwidth) will do.

C In either case, PD is preset to zero by the solver,

C so that only the nonzero elements need be loaded by JAC.

C Each call to JAC is preceded by a call to F with the same

C arguments NEQ, T, and Y. Thus to gain some efficiency,

C intermediate quantities shared by both calculations may be

C saved in a user COMMON block by F and not recomputed by JAC,

C if desired. Also, JAC may alter the Y array, if desired.

C JAC must be declared external in the calling program.

C Subroutine JAC may access user-defined real and integer

C work arrays, RPAR and IPAR, whose dimensions are set by the

C user in the main program.

C

C MF = The method flag. Used only for input. The legal values of

C MF are 10, 11, 12, 13, 14, 15, 20, 21, 22, 23, 24, 25,

C -11, -12, -14, -15, -21, -22, -24, -25.

C MF is a signed two-digit integer, MF = JSV\*(10\*METH + MITER).

C JSV = SIGN(MF) indicates the Jacobian-saving strategy..

C JSV = 1 means a copy of the Jacobian is saved for reuse

C in the corrector iteration algorithm.

C JSV = -1 means a copy of the Jacobian is not saved

C (valid only for MITER = 1, 2, 4, or 5).

C METH indicates the basic linear multistep method..

C METH = 1 means the implicit Adams method.

C METH = 2 means the method based on backward

C differentiation formulas (BDF-s).

C MITER indicates the corrector iteration method..

C MITER = 0 means functional iteration (no Jacobian matrix

C is involved).

C MITER = 1 means chord iteration with a user-supplied

C full (NEQ by NEQ) Jacobian.

C MITER = 2 means chord iteration with an internally

C generated (difference quotient) full Jacobian

C (using NEQ extra calls to F per df/dy value).

C MITER = 3 means chord iteration with an internally

C generated diagonal Jacobian approximation

C (using 1 extra call to F per df/dy evaluation).

C MITER = 4 means chord iteration with a user-supplied

C banded Jacobian.

C MITER = 5 means chord iteration with an internally

C generated banded Jacobian (using ML+MU+1 extra

C calls to F per df/dy evaluation).

C If MITER = 1 or 4, the user must supply a subroutine JAC

C (the name is arbitrary) as described above under JAC.

C For other values of MITER, a dummy argument can be used.

C

C RPAR User-specified array used to communicate real parameters

C to user-supplied subroutines. If RPAR is a vector, then

C it must be dimensioned in the user's main program. If it

C is unused or it is a scalar, then it need not be

C dimensioned.

C

C IPAR User-specified array used to communicate integer parameter

C to user-supplied subroutines. The comments on dimensioning

C RPAR apply to IPAR.

C-----------------------------------------------------------------------

C Optional Input.

C

C The following is a list of the optional input provided for in the

C call sequence. (See also Part ii.) For each such input variable,

C this table lists its name as used in this documentation, its

C location in the call sequence, its meaning, and the default value.

C The use of any of this input requires IOPT = 1, and in that

C case all of this input is examined. A value of zero for any

C of these optional input variables will cause the default value to be

C used. Thus to use a subset of the optional input, simply preload

C locations 5 to 10 in RWORK and IWORK to 0.0 and 0 respectively, and

C then set those of interest to nonzero values.

C

C NAME LOCATION MEANING AND DEFAULT VALUE

C

C H0 RWORK(5) The step size to be attempted on the first step.

C The default value is determined by the solver.

C

C HMAX RWORK(6) The maximum absolute step size allowed.

C The default value is infinite.

C

C HMIN RWORK(7) The minimum absolute step size allowed.

C The default value is 0. (This lower bound is not

C enforced on the final step before reaching TCRIT

C when ITASK = 4 or 5.)

C

C MAXORD IWORK(5) The maximum order to be allowed. The default

C value is 12 if METH = 1, and 5 if METH = 2.

C If MAXORD exceeds the default value, it will

C be reduced to the default value.

C If MAXORD is changed during the problem, it may

C cause the current order to be reduced.

C

C MXSTEP IWORK(6) Maximum number of (internally defined) steps

C allowed during one call to the solver.

C The default value is 500.

C

C MXHNIL IWORK(7) Maximum number of messages printed (per problem)

C warning that T + H = T on a step (H = step size).

C This must be positive to result in a non-default

C value. The default value is 10.

C

C-----------------------------------------------------------------------

C Optional Output.

C

C As optional additional output from DVODE, the variables listed

C below are quantities related to the performance of DVODE

C which are available to the user. These are communicated by way of

C the work arrays, but also have internal mnemonic names as shown.

C Except where stated otherwise, all of this output is defined

C on any successful return from DVODE, and on any return with

C ISTATE = -1, -2, -4, -5, or -6. On an illegal input return

C (ISTATE = -3), they will be unchanged from their existing values

C (if any), except possibly for TOLSF, LENRW, and LENIW.

C On any error return, output relevant to the error will be defined,

C as noted below.

C

C NAME LOCATION MEANING

C

C HU RWORK(11) The step size in t last used (successfully).

C

C HCUR RWORK(12) The step size to be attempted on the next step.

C

C TCUR RWORK(13) The current value of the independent variable

C which the solver has actually reached, i.e. the

C current internal mesh point in t. In the output,

C TCUR will always be at least as far from the

C initial value of t as the current argument T,

C but may be farther (if interpolation was done).

C

C TOLSF RWORK(14) A tolerance scale factor, greater than 1.0,

C computed when a request for too much accuracy was

C detected (ISTATE = -3 if detected at the start of

C the problem, ISTATE = -2 otherwise). If ITOL is

C left unaltered but RTOL and ATOL are uniformly

C scaled up by a factor of TOLSF for the next call,

C then the solver is deemed likely to succeed.

C (The user may also ignore TOLSF and alter the

C tolerance parameters in any other way appropriate.)

C

C NST IWORK(11) The number of steps taken for the problem so far.

C

C NFE IWORK(12) The number of f evaluations for the problem so far.

C

C NJE IWORK(13) The number of Jacobian evaluations so far.

C

C NQU IWORK(14) The method order last used (successfully).

C

C NQCUR IWORK(15) The order to be attempted on the next step.

C

C IMXER IWORK(16) The index of the component of largest magnitude in

C the weighted local error vector ( e(i)/EWT(i) ),

C on an error return with ISTATE = -4 or -5.

C

C LENRW IWORK(17) The length of RWORK actually required.

C This is defined on normal returns and on an illegal

C input return for insufficient storage.

C

C LENIW IWORK(18) The length of IWORK actually required.

C This is defined on normal returns and on an illegal

C input return for insufficient storage.

C

C NLU IWORK(19) The number of matrix LU decompositions so far.

C

C NNI IWORK(20) The number of nonlinear (Newton) iterations so far.

C

C NCFN IWORK(21) The number of convergence failures of the nonlinear

C solver so far.

C

C NETF IWORK(22) The number of error test failures of the integrator

C so far.

C

C The following two arrays are segments of the RWORK array which

C may also be of interest to the user as optional output.

C For each array, the table below gives its internal name,

C its base address in RWORK, and its description.

C

C NAME BASE ADDRESS DESCRIPTION

C

C YH 21 The Nordsieck history array, of size NYH by

C (NQCUR + 1), where NYH is the initial value

C of NEQ. For j = 0,1,...,NQCUR, column j+1

C of YH contains HCUR\*\*j/factorial(j) times

C the j-th derivative of the interpolating

C polynomial currently representing the

C solution, evaluated at t = TCUR.

C

C ACOR LENRW-NEQ+1 Array of size NEQ used for the accumulated

C corrections on each step, scaled in the output

C to represent the estimated local error in Y

C on the last step. This is the vector e in

C the description of the error control. It is

C defined only on a successful return from DVODE.

C

C-----------------------------------------------------------------------

C Interrupting and Restarting

C

C If the integration of a given problem by DVODE is to be

C interrrupted and then later continued, such as when restarting

C an interrupted run or alternating between two or more ODE problems,

C the user should save, following the return from the last DVODE call

C prior to the interruption, the contents of the call sequence

C variables and internal COMMON blocks, and later restore these

C values before the next DVODE call for that problem. To save

C and restore the COMMON blocks, use subroutine DVSRCO, as

C described below in part ii.

C

C In addition, if non-default values for either LUN or MFLAG are

C desired, an extra call to XSETUN and/or XSETF should be made just

C before continuing the integration. See Part ii below for details.

C

C-----------------------------------------------------------------------

C Part ii. Other Routines Callable.

C

C The following are optional calls which the user may make to

C gain additional capabilities in conjunction with DVODE.

C (The routines XSETUN and XSETF are designed to conform to the

C SLATEC error handling package.)

C

C FORM OF CALL FUNCTION

C CALL XSETUN(LUN) Set the logical unit number, LUN, for

C output of messages from DVODE, if

C the default is not desired.

C The default value of LUN is 6.

C

C CALL XSETF(MFLAG) Set a flag to control the printing of

C messages by DVODE.

C MFLAG = 0 means do not print. (Danger..

C This risks losing valuable information.)

C MFLAG = 1 means print (the default).

C

C Either of the above calls may be made at

C any time and will take effect immediately.

C

C CALL DVSRCO(RSAV,ISAV,JOB) Saves and restores the contents of

C the internal COMMON blocks used by

C DVODE. (See Part iii below.)

C RSAV must be a real array of length 49

C or more, and ISAV must be an integer

C array of length 40 or more.

C JOB=1 means save COMMON into RSAV/ISAV.

C JOB=2 means restore COMMON from RSAV/ISAV.

C DVSRCO is useful if one is

C interrupting a run and restarting

C later, or alternating between two or

C more problems solved with DVODE.

C

C CALL DVINDY(,,,,,) Provide derivatives of y, of various

C (See below.) orders, at a specified point T, if

C desired. It may be called only after

C a successful return from DVODE.

C

C The detailed instructions for using DVINDY are as follows.

C The form of the call is..

C

C CALL DVINDY (T, K, RWORK(21), NYH, DKY, IFLAG)

C

C The input parameters are..

C

C T = Value of independent variable where answers are desired

C (normally the same as the T last returned by DVODE).

C For valid results, T must lie between TCUR - HU and TCUR.

C (See optional output for TCUR and HU.)

C K = Integer order of the derivative desired. K must satisfy

C 0 .le. K .le. NQCUR, where NQCUR is the current order

C (see optional output). The capability corresponding

C to K = 0, i.e. computing y(T), is already provided

C by DVODE directly. Since NQCUR .ge. 1, the first

C derivative dy/dt is always available with DVINDY.

C RWORK(21) = The base address of the history array YH.

C NYH = Column length of YH, equal to the initial value of NEQ.

C

C The output parameters are..

C

C DKY = A real array of length NEQ containing the computed value

C of the K-th derivative of y(t).

C IFLAG = Integer flag, returned as 0 if K and T were legal,

C -1 if K was illegal, and -2 if T was illegal.

C On an error return, a message is also written.

C-----------------------------------------------------------------------

C Part iii. COMMON Blocks.

C If DVODE is to be used in an overlay situation, the user

C must declare, in the primary overlay, the variables in..

C (1) the call sequence to DVODE,

C (2) the two internal COMMON blocks

C /DVOD01/ of length 81 (48 double precision words

C followed by 33 integer words),

C /DVOD02/ of length 9 (1 double precision word

C followed by 8 integer words),

C

C If DVODE is used on a system in which the contents of internal

C COMMON blocks are not preserved between calls, the user should

C declare the above two COMMON blocks in his main program to insure

C that their contents are preserved.

C

C-----------------------------------------------------------------------

C Part iv. Optionally Replaceable Solver Routines.

C

C Below are descriptions of two routines in the DVODE package which

C relate to the measurement of errors. Either routine can be

C replaced by a user-supplied version, if desired. However, since such

C a replacement may have a major impact on performance, it should be

C done only when absolutely necessary, and only with great caution.

C (Note.. The means by which the package version of a routine is

C superseded by the user's version may be system-dependent.)

C

C (a) DEWSET.

C The following subroutine is called just before each internal

C integration step, and sets the array of error weights, EWT, as

C described under ITOL/RTOL/ATOL above..

C SUBROUTINE DEWSET (NEQ, ITOL, RTOL, ATOL, YCUR, EWT)

C where NEQ, ITOL, RTOL, and ATOL are as in the DVODE call sequence,

C YCUR contains the current dependent variable vector, and

C EWT is the array of weights set by DEWSET.

C

C If the user supplies this subroutine, it must return in EWT(i)

C (i = 1,...,NEQ) a positive quantity suitable for comparison with

C errors in Y(i). The EWT array returned by DEWSET is passed to the

C DVNORM routine (See below.), and also used by DVODE in the computation

C of the optional output IMXER, the diagonal Jacobian approximation,

C and the increments for difference quotient Jacobians.

C

C In the user-supplied version of DEWSET, it may be desirable to use

C the current values of derivatives of y. Derivatives up to order NQ

C are available from the history array YH, described above under

C Optional Output. In DEWSET, YH is identical to the YCUR array,

C extended to NQ + 1 columns with a column length of NYH and scale

C factors of h\*\*j/factorial(j). On the first call for the problem,

C given by NST = 0, NQ is 1 and H is temporarily set to 1.0.

C NYH is the initial value of NEQ. The quantities NQ, H, and NST

C can be obtained by including in DEWSET the statements..

C DOUBLE PRECISION RVOD, H, HU

C COMMON /DVOD01/ RVOD(48), IVOD(33)

C COMMON /DVOD02/ HU, NCFN, NETF, NFE, NJE, NLU, NNI, NQU, NST

C NQ = IVOD(28)

C H = RVOD(21)

C Thus, for example, the current value of dy/dt can be obtained as

C YCUR(NYH+i)/H (i=1,...,NEQ) (and the division by H is

C unnecessary when NST = 0).

C

C (b) DVNORM.

C The following is a real function routine which computes the weighted

C root-mean-square norm of a vector v..

C D = DVNORM (N, V, W)

C where..

C N = the length of the vector,

C V = real array of length N containing the vector,

C W = real array of length N containing weights,

C D = sqrt( (1/N) \* sum(V(i)\*W(i))\*\*2 ).

C DVNORM is called with N = NEQ and with W(i) = 1.0/EWT(i), where

C EWT is as set by subroutine DEWSET.

C

C If the user supplies this function, it should return a non-negative

C value of DVNORM suitable for use in the error control in DVODE.

C None of the arguments should be altered by DVNORM.

C For example, a user-supplied DVNORM routine might..

C -substitute a max-norm of (V(i)\*W(i)) for the rms-norm, or

C -ignore some components of V in the norm, with the effect of

C suppressing the error control on those components of Y.

C-----------------------------------------------------------------------

C Other Routines in the DVODE Package.

C

C In addition to subroutine DVODE, the DVODE package includes the

C following subroutines and function routines..

C DVHIN computes an approximate step size for the initial step.

C DVINDY computes an interpolated value of the y vector at t = TOUT.

C DVSTEP is the core integrator, which does one step of the

C integration and the associated error control.

C DVSET sets all method coefficients and test constants.

C DVNLSD solves the underlying nonlinear system -- the corrector.

C DVJAC computes and preprocesses the Jacobian matrix J = df/dy

C and the Newton iteration matrix P = I - (h/l1)\*J.

C DVSOL manages solution of linear system in chord iteration.

C DVJUST adjusts the history array on a change of order.

C DEWSET sets the error weight vector EWT before each step.

C DVNORM computes the weighted r.m.s. norm of a vector.

C DVSRCO is a user-callable routine to save and restore

C the contents of the internal COMMON blocks.

C DACOPY is a routine to copy one two-dimensional array to another.

C DGEFA and DGESL are routines from LINPACK for solving full

C systems of linear algebraic equations.

C DGBFA and DGBSL are routines from LINPACK for solving banded

C linear systems.

C DAXPY, DSCAL, and DCOPY are basic linear algebra modules (BLAS).

C D1MACH sets the unit roundoff of the machine.

C XERRWD, XSETUN, XSETF, and IXSAV handle the printing of all

C error messages and warnings. XERRWD is machine-dependent.

C Note.. DVNORM, D1MACH, and IXSAV are function routines.

C All the others are subroutines.

C

C The intrinsic and external routines used by the DVODE package are..

C ABS, MAX, MIN, REAL, SIGN, SQRT, and WRITE.

C

C-----------------------------------------------------------------------

C

C Type declarations for labeled COMMON block DVOD01 --------------------

C

DOUBLE PRECISION ACNRM, CCMXJ, CONP, CRATE, DRC, EL,

1 ETA, ETAMAX, H, HMIN, HMXI, HNEW, HSCAL, PRL1,

2 RC, RL1, TAU, TQ, TN, UROUND

INTEGER ICF, INIT, IPUP, JCUR, JSTART, JSV, KFLAG, KUTH,

1 L, LMAX, LYH, LEWT, LACOR, LSAVF, LWM, LIWM,

2 LOCJS, MAXORD, METH, MITER, MSBJ, MXHNIL, MXSTEP,

3 N, NEWH, NEWQ, NHNIL, NQ, NQNYH, NQWAIT, NSLJ,

4 NSLP, NYH

C

C Type declarations for labeled COMMON block DVOD02 --------------------

C

DOUBLE PRECISION HU

INTEGER NCFN, NETF, NFE, NJE, NLU, NNI, NQU, NST

C

C Type declarations for local variables --------------------------------

C

EXTERNAL DVNLSD

LOGICAL IHIT

DOUBLE PRECISION ATOLI, BIG, EWTI, FOUR, H0, HMAX, HMX, HUN, ONE,

1 PT2, RH, RTOLI, SIZE, TCRIT, TNEXT, TOLSF, TP, TWO, ZERO

INTEGER I, IER, IFLAG, IMXER, JCO, KGO, LENIW, LENJ, LENP, LENRW,

1 LENWM, LF0, MBAND, MFA, ML, MORD, MU, MXHNL0, MXSTP0, NITER,

2 NSLAST

CHARACTER\*80 MSG

C

C Type declaration for function subroutines called ---------------------

C

DOUBLE PRECISION D1MACH, DVNORM

C

DIMENSION MORD(2)

C-----------------------------------------------------------------------

C The following Fortran-77 declaration is to cause the values of the

C listed (local) variables to be saved between calls to DVODE.

C-----------------------------------------------------------------------

SAVE MORD, MXHNL0, MXSTP0

SAVE ZERO, ONE, TWO, FOUR, PT2, HUN

C-----------------------------------------------------------------------

C The following internal COMMON blocks contain variables which are

C communicated between subroutines in the DVODE package, or which are

C to be saved between calls to DVODE.

C In each block, real variables precede integers.

C The block /DVOD01/ appears in subroutines DVODE, DVINDY, DVSTEP,

C DVSET, DVNLSD, DVJAC, DVSOL, DVJUST and DVSRCO.

C The block /DVOD02/ appears in subroutines DVODE, DVINDY, DVSTEP,

C DVNLSD, DVJAC, and DVSRCO.

C

C The variables stored in the internal COMMON blocks are as follows..

C

C ACNRM = Weighted r.m.s. norm of accumulated correction vectors.

C CCMXJ = Threshhold on DRC for updating the Jacobian. (See DRC.)

C CONP = The saved value of TQ(5).

C CRATE = Estimated corrector convergence rate constant.

C DRC = Relative change in H\*RL1 since last DVJAC call.

C EL = Real array of integration coefficients. See DVSET.

C ETA = Saved tentative ratio of new to old H.

C ETAMAX = Saved maximum value of ETA to be allowed.

C H = The step size.

C HMIN = The minimum absolute value of the step size H to be used.

C HMXI = Inverse of the maximum absolute value of H to be used.

C HMXI = 0.0 is allowed and corresponds to an infinite HMAX.

C HNEW = The step size to be attempted on the next step.

C HSCAL = Stepsize in scaling of YH array.

C PRL1 = The saved value of RL1.

C RC = Ratio of current H\*RL1 to value on last DVJAC call.

C RL1 = The reciprocal of the coefficient EL(1).

C TAU = Real vector of past NQ step sizes, length 13.

C TQ = A real vector of length 5 in which DVSET stores constants

C used for the convergence test, the error test, and the

C selection of H at a new order.

C TN = The independent variable, updated on each step taken.

C UROUND = The machine unit roundoff. The smallest positive real number

C such that 1.0 + UROUND .ne. 1.0

C ICF = Integer flag for convergence failure in DVNLSD..

C 0 means no failures.

C 1 means convergence failure with out of date Jacobian

C (recoverable error).

C 2 means convergence failure with current Jacobian or

C singular matrix (unrecoverable error).

C INIT = Saved integer flag indicating whether initialization of the

C problem has been done (INIT = 1) or not.

C IPUP = Saved flag to signal updating of Newton matrix.

C JCUR = Output flag from DVJAC showing Jacobian status..

C JCUR = 0 means J is not current.

C JCUR = 1 means J is current.

C JSTART = Integer flag used as input to DVSTEP..

C 0 means perform the first step.

C 1 means take a new step continuing from the last.

C -1 means take the next step with a new value of MAXORD,

C HMIN, HMXI, N, METH, MITER, and/or matrix parameters.

C On return, DVSTEP sets JSTART = 1.

C JSV = Integer flag for Jacobian saving, = sign(MF).

C KFLAG = A completion code from DVSTEP with the following meanings..

C 0 the step was succesful.

C -1 the requested error could not be achieved.

C -2 corrector convergence could not be achieved.

C -3, -4 fatal error in VNLS (can not occur here).

C KUTH = Input flag to DVSTEP showing whether H was reduced by the

C driver. KUTH = 1 if H was reduced, = 0 otherwise.

C L = Integer variable, NQ + 1, current order plus one.

C LMAX = MAXORD + 1 (used for dimensioning).

C LOCJS = A pointer to the saved Jacobian, whose storage starts at

C WM(LOCJS), if JSV = 1.

C LYH, LEWT, LACOR, LSAVF, LWM, LIWM = Saved integer pointers

C to segments of RWORK and IWORK.

C MAXORD = The maximum order of integration method to be allowed.

C METH/MITER = The method flags. See MF.

C MSBJ = The maximum number of steps between J evaluations, = 50.

C MXHNIL = Saved value of optional input MXHNIL.

C MXSTEP = Saved value of optional input MXSTEP.

C N = The number of first-order ODEs, = NEQ.

C NEWH = Saved integer to flag change of H.

C NEWQ = The method order to be used on the next step.

C NHNIL = Saved counter for occurrences of T + H = T.

C NQ = Integer variable, the current integration method order.

C NQNYH = Saved value of NQ\*NYH.

C NQWAIT = A counter controlling the frequency of order changes.

C An order change is about to be considered if NQWAIT = 1.

C NSLJ = The number of steps taken as of the last Jacobian update.

C NSLP = Saved value of NST as of last Newton matrix update.

C NYH = Saved value of the initial value of NEQ.

C HU = The step size in t last used.

C NCFN = Number of nonlinear convergence failures so far.

C NETF = The number of error test failures of the integrator so far.

C NFE = The number of f evaluations for the problem so far.

C NJE = The number of Jacobian evaluations so far.

C NLU = The number of matrix LU decompositions so far.

C NNI = Number of nonlinear iterations so far.

C NQU = The method order last used.

C NST = The number of steps taken for the problem so far.

C-----------------------------------------------------------------------

COMMON /DVOD01/ ACNRM, CCMXJ, CONP, CRATE, DRC, EL(13),

1 ETA, ETAMAX, H, HMIN, HMXI, HNEW, HSCAL, PRL1,

2 RC, RL1, TAU(13), TQ(5), TN, UROUND,

3 ICF, INIT, IPUP, JCUR, JSTART, JSV, KFLAG, KUTH,

4 L, LMAX, LYH, LEWT, LACOR, LSAVF, LWM, LIWM,

5 LOCJS, MAXORD, METH, MITER, MSBJ, MXHNIL, MXSTEP,

6 N, NEWH, NEWQ, NHNIL, NQ, NQNYH, NQWAIT, NSLJ,

7 NSLP, NYH

COMMON /DVOD02/ HU, NCFN, NETF, NFE, NJE, NLU, NNI, NQU, NST

C

DATA MORD(1) /12/, MORD(2) /5/, MXSTP0 /5000/, MXHNL0 /10/

DATA ZERO /0.0D0/, ONE /1.0D0/, TWO /2.0D0/, FOUR /4.0D0/,

1 PT2 /0.2D0/, HUN /100.0D0/

C-----------------------------------------------------------------------

C Block A.

C This code block is executed on every call.

C It tests ISTATE and ITASK for legality and branches appropriately.

C If ISTATE .gt. 1 but the flag INIT shows that initialization has

C not yet been done, an error return occurs.

C If ISTATE = 1 and TOUT = T, return immediately.

C-----------------------------------------------------------------------

IF (ISTATE .LT. 1 .OR. ISTATE .GT. 3) GO TO 601

IF (ITASK .LT. 1 .OR. ITASK .GT. 5) GO TO 602

IF (ISTATE .EQ. 1) GO TO 10

IF (INIT .NE. 1) GO TO 603

IF (ISTATE .EQ. 2) GO TO 200

GO TO 20

10 INIT = 0

IF (TOUT .EQ. T) RETURN

C-----------------------------------------------------------------------

C Block B.

C The next code block is executed for the initial call (ISTATE = 1),

C or for a continuation call with parameter changes (ISTATE = 3).

C It contains checking of all input and various initializations.

C

C First check legality of the non-optional input NEQ, ITOL, IOPT,

C MF, ML, and MU.

C-----------------------------------------------------------------------

20 IF (NEQ .LE. 0) GO TO 604

IF (ISTATE .EQ. 1) GO TO 25

IF (NEQ .GT. N) GO TO 605

25 N = NEQ

IF (ITOL .LT. 1 .OR. ITOL .GT. 4) GO TO 606

IF (IOPT .LT. 0 .OR. IOPT .GT. 1) GO TO 607

JSV = SIGN(1,MF)

MFA = ABS(MF)

METH = MFA/10

MITER = MFA - 10\*METH

IF (METH .LT. 1 .OR. METH .GT. 2) GO TO 608

IF (MITER .LT. 0 .OR. MITER .GT. 5) GO TO 608

IF (MITER .LE. 3) GO TO 30

ML = IWORK(1)

MU = IWORK(2)

IF (ML .LT. 0 .OR. ML .GE. N) GO TO 609

IF (MU .LT. 0 .OR. MU .GE. N) GO TO 610

30 CONTINUE

C Next process and check the optional input. ---------------------------

IF (IOPT .EQ. 1) GO TO 40

MAXORD = MORD(METH)

MXSTEP = MXSTP0

MXHNIL = MXHNL0

IF (ISTATE .EQ. 1) H0 = ZERO

HMXI = ZERO

HMIN = ZERO

GO TO 60

40 MAXORD = IWORK(5)

IF (MAXORD .LT. 0) GO TO 611

IF (MAXORD .EQ. 0) MAXORD = 100

MAXORD = MIN(MAXORD,MORD(METH))

MXSTEP = IWORK(6)

IF (MXSTEP .LT. 0) GO TO 612

IF (MXSTEP .EQ. 0) MXSTEP = MXSTP0

MXHNIL = IWORK(7)

IF (MXHNIL .LT. 0) GO TO 613

IF (MXHNIL .EQ. 0) MXHNIL = MXHNL0

IF (ISTATE .NE. 1) GO TO 50

H0 = RWORK(5)

IF ((TOUT - T)\*H0 .LT. ZERO) GO TO 614

50 HMAX = RWORK(6)

IF (HMAX .LT. ZERO) GO TO 615

HMXI = ZERO

IF (HMAX .GT. ZERO) HMXI = ONE/HMAX

HMIN = RWORK(7)

IF (HMIN .LT. ZERO) GO TO 616

C-----------------------------------------------------------------------

C Set work array pointers and check lengths LRW and LIW.

C Pointers to segments of RWORK and IWORK are named by prefixing L to

C the name of the segment. E.g., the segment YH starts at RWORK(LYH).

C Segments of RWORK (in order) are denoted YH, WM, EWT, SAVF, ACOR.

C Within WM, LOCJS is the location of the saved Jacobian (JSV .gt. 0).

C-----------------------------------------------------------------------

60 LYH = 21

IF (ISTATE .EQ. 1) NYH = N

LWM = LYH + (MAXORD + 1)\*NYH

JCO = MAX(0,JSV)

IF (MITER .EQ. 0) LENWM = 0

IF (MITER .EQ. 1 .OR. MITER .EQ. 2) THEN

LENWM = 2 + (1 + JCO)\*N\*N

LOCJS = N\*N + 3

ENDIF

IF (MITER .EQ. 3) LENWM = 2 + N

IF (MITER .EQ. 4 .OR. MITER .EQ. 5) THEN

MBAND = ML + MU + 1

LENP = (MBAND + ML)\*N

LENJ = MBAND\*N

LENWM = 2 + LENP + JCO\*LENJ

LOCJS = LENP + 3

ENDIF

LEWT = LWM + LENWM

LSAVF = LEWT + N

LACOR = LSAVF + N

LENRW = LACOR + N - 1

IWORK(17) = LENRW

LIWM = 1

LENIW = 30 + N

IF (MITER .EQ. 0 .OR. MITER .EQ. 3) LENIW = 30

IWORK(18) = LENIW

IF (LENRW .GT. LRW) GO TO 617

IF (LENIW .GT. LIW) GO TO 618

C Check RTOL and ATOL for legality. ------------------------------------

RTOLI = RTOL(1)

ATOLI = ATOL(1)

DO 70 I = 1,N

IF (ITOL .GE. 3) RTOLI = RTOL(I)

IF (ITOL .EQ. 2 .OR. ITOL .EQ. 4) ATOLI = ATOL(I)

IF (RTOLI .LT. ZERO) GO TO 619

IF (ATOLI .LT. ZERO) GO TO 620

70 CONTINUE

IF (ISTATE .EQ. 1) GO TO 100

C If ISTATE = 3, set flag to signal parameter changes to DVSTEP. -------

JSTART = -1

IF (NQ .LE. MAXORD) GO TO 90

C MAXORD was reduced below NQ. Copy YH(\*,MAXORD+2) into SAVF. ---------

CALL DCOPY (N, RWORK(LWM), 1, RWORK(LSAVF), 1)

C Reload WM(1) = RWORK(LWM), since LWM may have changed. ---------------

90 IF (MITER .GT. 0) RWORK(LWM) = SQRT(UROUND)

C bug fix 12 Nov 1998

GO TO 200

C-----------------------------------------------------------------------

C Block C.

C The next block is for the initial call only (ISTATE = 1).

C It contains all remaining initializations, the initial call to F,

C and the calculation of the initial step size.

C The error weights in EWT are inverted after being loaded.

C-----------------------------------------------------------------------

100 UROUND = D1MACH(4)

TN = T

IF (ITASK .NE. 4 .AND. ITASK .NE. 5) GO TO 110

TCRIT = RWORK(1)

IF ((TCRIT - TOUT)\*(TOUT - T) .LT. ZERO) GO TO 625

IF (H0 .NE. ZERO .AND. (T + H0 - TCRIT)\*H0 .GT. ZERO)

1 H0 = TCRIT - T

110 JSTART = 0

IF (MITER .GT. 0) RWORK(LWM) = SQRT(UROUND)

CCMXJ = PT2

MSBJ = 50

NHNIL = 0

NST = 0

NJE = 0

NNI = 0

NCFN = 0

NETF = 0

NLU = 0

NSLJ = 0

NSLAST = 0

HU = ZERO

NQU = 0

C Initial call to F. (LF0 points to YH(\*,2).) -------------------------

LF0 = LYH + NYH

CALL F (N, T, Y, RWORK(LF0), RPAR, IPAR)

NFE = 1

C Load the initial value vector in YH. ---------------------------------

CALL DCOPY (N, Y, 1, RWORK(LYH), 1)

C Load and invert the EWT array. (H is temporarily set to 1.0.) -------

NQ = 1

H = ONE

CALL DEWSET (N, ITOL, RTOL, ATOL, RWORK(LYH), RWORK(LEWT))

DO 120 I = 1,N

IF (RWORK(I+LEWT-1) .LE. ZERO) GO TO 621

120 RWORK(I+LEWT-1) = ONE/RWORK(I+LEWT-1)

IF (H0 .NE. ZERO) GO TO 180

C Call DVHIN to set initial step size H0 to be attempted. --------------

CALL DVHIN (N, T, RWORK(LYH), RWORK(LF0), F, RPAR, IPAR, TOUT,

1 UROUND, RWORK(LEWT), ITOL, ATOL, Y, RWORK(LACOR), H0,

2 NITER, IER)

NFE = NFE + NITER

IF (IER .NE. 0) GO TO 622

C Adjust H0 if necessary to meet HMAX bound. ---------------------------

180 RH = ABS(H0)\*HMXI

IF (RH .GT. ONE) H0 = H0/RH

C Load H with H0 and scale YH(\*,2) by H0. ------------------------------

H = H0

CALL DSCAL (N, H0, RWORK(LF0), 1)

GO TO 270

C-----------------------------------------------------------------------

C Block D.

C The next code block is for continuation calls only (ISTATE = 2 or 3)

C and is to check stop conditions before taking a step.

C-----------------------------------------------------------------------

200 NSLAST = NST

KUTH = 0

GO TO (210, 250, 220, 230, 240), ITASK

210 IF ((TN - TOUT)\*H .LT. ZERO) GO TO 250

CALL DVINDY (TOUT, 0, RWORK(LYH), NYH, Y, IFLAG)

IF (IFLAG .NE. 0) GO TO 627

T = TOUT

GO TO 420

220 TP = TN - HU\*(ONE + HUN\*UROUND)

IF ((TP - TOUT)\*H .GT. ZERO) GO TO 623

IF ((TN - TOUT)\*H .LT. ZERO) GO TO 250

GO TO 400

230 TCRIT = RWORK(1)

IF ((TN - TCRIT)\*H .GT. ZERO) GO TO 624

IF ((TCRIT - TOUT)\*H .LT. ZERO) GO TO 625

IF ((TN - TOUT)\*H .LT. ZERO) GO TO 245

CALL DVINDY (TOUT, 0, RWORK(LYH), NYH, Y, IFLAG)

IF (IFLAG .NE. 0) GO TO 627

T = TOUT

GO TO 420

240 TCRIT = RWORK(1)

IF ((TN - TCRIT)\*H .GT. ZERO) GO TO 624

245 HMX = ABS(TN) + ABS(H)

IHIT = ABS(TN - TCRIT) .LE. HUN\*UROUND\*HMX

IF (IHIT) GO TO 400

TNEXT = TN + HNEW\*(ONE + FOUR\*UROUND)

IF ((TNEXT - TCRIT)\*H .LE. ZERO) GO TO 250

H = (TCRIT - TN)\*(ONE - FOUR\*UROUND)

KUTH = 1

C-----------------------------------------------------------------------

C Block E.

C The next block is normally executed for all calls and contains

C the call to the one-step core integrator DVSTEP.

C

C This is a looping point for the integration steps.

C

C First check for too many steps being taken, update EWT (if not at

C start of problem), check for too much accuracy being requested, and

C check for H below the roundoff level in T.

C-----------------------------------------------------------------------

250 CONTINUE

IF ((NST-NSLAST) .GE. MXSTEP) GO TO 500

CALL DEWSET (N, ITOL, RTOL, ATOL, RWORK(LYH), RWORK(LEWT))

DO 260 I = 1,N

IF (RWORK(I+LEWT-1) .LE. ZERO) GO TO 510

260 RWORK(I+LEWT-1) = ONE/RWORK(I+LEWT-1)

270 TOLSF = UROUND\*DVNORM (N, RWORK(LYH), RWORK(LEWT))

IF (TOLSF .LE. ONE) GO TO 280

TOLSF = TOLSF\*TWO

IF (NST .EQ. 0) GO TO 626

GO TO 520

280 IF ((TN + H) .NE. TN) GO TO 290

NHNIL = NHNIL + 1

IF (NHNIL .GT. MXHNIL) GO TO 290

MSG = 'DVODE-- Warning..internal T (=R1) and H (=R2) are'

CALL XERRWD (MSG, 50, 101, 1, 0, 0, 0, 0, ZERO, ZERO)

MSG=' such that in the machine, T + H = T on the next step '

CALL XERRWD (MSG, 60, 101, 1, 0, 0, 0, 0, ZERO, ZERO)

MSG = ' (H = step size). solver will continue anyway'

CALL XERRWD (MSG, 50, 101, 1, 0, 0, 0, 2, TN, H)

IF (NHNIL .LT. MXHNIL) GO TO 290

MSG = 'DVODE-- Above warning has been issued I1 times. '

CALL XERRWD (MSG, 50, 102, 1, 0, 0, 0, 0, ZERO, ZERO)

MSG = ' it will not be issued again for this problem'

CALL XERRWD (MSG, 50, 102, 1, 1, MXHNIL, 0, 0, ZERO, ZERO)

290 CONTINUE

C-----------------------------------------------------------------------

C CALL DVSTEP (Y, YH, NYH, YH, EWT, SAVF, VSAV, ACOR,

C WM, IWM, F, JAC, F, DVNLSD, RPAR, IPAR)

C-----------------------------------------------------------------------

CALL DVSTEP (Y, RWORK(LYH), NYH, RWORK(LYH), RWORK(LEWT),

1 RWORK(LSAVF), Y, RWORK(LACOR), RWORK(LWM), IWORK(LIWM),

2 F, JAC, F, DVNLSD, RPAR, IPAR)

KGO = 1 - KFLAG

C Branch on KFLAG. Note..In this version, KFLAG can not be set to -3.

C KFLAG .eq. 0, -1, -2

GO TO (300, 530, 540), KGO

C-----------------------------------------------------------------------

C Block F.

C The following block handles the case of a successful return from the

C core integrator (KFLAG = 0). Test for stop conditions.

C-----------------------------------------------------------------------

300 INIT = 1

KUTH = 0

GO TO (310, 400, 330, 340, 350), ITASK

C ITASK = 1. If TOUT has been reached, interpolate. -------------------

310 IF ((TN - TOUT)\*H .LT. ZERO) GO TO 250

CALL DVINDY (TOUT, 0, RWORK(LYH), NYH, Y, IFLAG)

T = TOUT

GO TO 420

C ITASK = 3. Jump to exit if TOUT was reached. ------------------------

330 IF ((TN - TOUT)\*H .GE. ZERO) GO TO 400

GO TO 250

C ITASK = 4. See if TOUT or TCRIT was reached. Adjust H if necessary.

340 IF ((TN - TOUT)\*H .LT. ZERO) GO TO 345

CALL DVINDY (TOUT, 0, RWORK(LYH), NYH, Y, IFLAG)

T = TOUT

GO TO 420

345 HMX = ABS(TN) + ABS(H)

IHIT = ABS(TN - TCRIT) .LE. HUN\*UROUND\*HMX

IF (IHIT) GO TO 400

TNEXT = TN + HNEW\*(ONE + FOUR\*UROUND)

IF ((TNEXT - TCRIT)\*H .LE. ZERO) GO TO 250

H = (TCRIT - TN)\*(ONE - FOUR\*UROUND)

KUTH = 1

GO TO 250

C ITASK = 5. See if TCRIT was reached and jump to exit. ---------------

350 HMX = ABS(TN) + ABS(H)

IHIT = ABS(TN - TCRIT) .LE. HUN\*UROUND\*HMX

C-----------------------------------------------------------------------

C Block G.

C The following block handles all successful returns from DVODE.

C If ITASK .ne. 1, Y is loaded from YH and T is set accordingly.

C ISTATE is set to 2, and the optional output is loaded into the work

C arrays before returning.

C-----------------------------------------------------------------------

400 CONTINUE

CALL DCOPY (N, RWORK(LYH), 1, Y, 1)

T = TN

IF (ITASK .NE. 4 .AND. ITASK .NE. 5) GO TO 420

IF (IHIT) T = TCRIT

420 ISTATE = 2

RWORK(11) = HU

RWORK(12) = HNEW

RWORK(13) = TN

IWORK(11) = NST

IWORK(12) = NFE

IWORK(13) = NJE

IWORK(14) = NQU

IWORK(15) = NEWQ

IWORK(19) = NLU

IWORK(20) = NNI

IWORK(21) = NCFN

IWORK(22) = NETF

RETURN

C-----------------------------------------------------------------------

C Block H.

C The following block handles all unsuccessful returns other than

C those for illegal input. First the error message routine is called.

C if there was an error test or convergence test failure, IMXER is set.

C Then Y is loaded from YH, and T is set to TN.

C The optional output is loaded into the work arrays before returning.

C-----------------------------------------------------------------------

C The maximum number of steps was taken before reaching TOUT. ----------

500 MSG = 'DVODE-- At current T (=R1), MXSTEP (=I1) steps '

CALL XERRWD (MSG, 50, 201, 1, 0, 0, 0, 0, ZERO, ZERO)

MSG = ' taken on this call before reaching TOUT '

CALL XERRWD (MSG, 50, 201, 1, 1, MXSTEP, 0, 1, TN, ZERO)

ISTATE = -1

GO TO 580

C EWT(i) .le. 0.0 for some i (not at start of problem). ----------------

510 EWTI = RWORK(LEWT+I-1)

MSG = 'DVODE-- At T (=R1), EWT(I1) has become R2 .le. 0.'

CALL XERRWD (MSG, 50, 202, 1, 1, I, 0, 2, TN, EWTI)

ISTATE = -6

GO TO 580

C Too much accuracy requested for machine precision. -------------------

520 MSG = 'DVODE-- At T (=R1), too much accuracy requested '

CALL XERRWD (MSG, 50, 203, 1, 0, 0, 0, 0, ZERO, ZERO)

MSG = ' for precision of machine.. see TOLSF (=R2) '

CALL XERRWD (MSG, 50, 203, 1, 0, 0, 0, 2, TN, TOLSF)

RWORK(14) = TOLSF

ISTATE = -2

GO TO 580

C KFLAG = -1. Error test failed repeatedly or with ABS(H) = HMIN. -----

530 MSG = 'DVODE-- At T(=R1) and step size H(=R2), the error'

CALL XERRWD (MSG, 50, 204, 1, 0, 0, 0, 0, ZERO, ZERO)

MSG = ' test failed repeatedly or with abs(H) = HMIN'

CALL XERRWD (MSG, 50, 204, 1, 0, 0, 0, 2, TN, H)

ISTATE = -4

GO TO 560

C KFLAG = -2. Convergence failed repeatedly or with ABS(H) = HMIN. ----

540 MSG = 'DVODE-- At T (=R1) and step size H (=R2), the '

CALL XERRWD (MSG, 50, 205, 1, 0, 0, 0, 0, ZERO, ZERO)

MSG = ' corrector convergence failed repeatedly '

CALL XERRWD (MSG, 50, 205, 1, 0, 0, 0, 0, ZERO, ZERO)

MSG = ' or with abs(H) = HMIN '

CALL XERRWD (MSG, 30, 205, 1, 0, 0, 0, 2, TN, H)

ISTATE = -5

C Compute IMXER if relevant. -------------------------------------------

560 BIG = ZERO

IMXER = 1

DO 570 I = 1,N

SIZE = ABS(RWORK(I+LACOR-1)\*RWORK(I+LEWT-1))

IF (BIG .GE. SIZE) GO TO 570

BIG = SIZE

IMXER = I

570 CONTINUE

IWORK(16) = IMXER

C Set Y vector, T, and optional output. --------------------------------

580 CONTINUE

CALL DCOPY (N, RWORK(LYH), 1, Y, 1)

T = TN

RWORK(11) = HU

RWORK(12) = H

RWORK(13) = TN

IWORK(11) = NST

IWORK(12) = NFE

IWORK(13) = NJE

IWORK(14) = NQU

IWORK(15) = NQ

IWORK(19) = NLU

IWORK(20) = NNI

IWORK(21) = NCFN

IWORK(22) = NETF

RETURN

C-----------------------------------------------------------------------

C Block I.

C The following block handles all error returns due to illegal input

C (ISTATE = -3), as detected before calling the core integrator.

C First the error message routine is called. If the illegal input

C is a negative ISTATE, the run is aborted (apparent infinite loop).

C-----------------------------------------------------------------------

601 MSG = 'DVODE-- ISTATE (=I1) illegal '

CALL XERRWD (MSG, 30, 1, 1, 1, ISTATE, 0, 0, ZERO, ZERO)

IF (ISTATE .LT. 0) GO TO 800

GO TO 700

602 MSG = 'DVODE-- ITASK (=I1) illegal '

CALL XERRWD (MSG, 30, 2, 1, 1, ITASK, 0, 0, ZERO, ZERO)

GO TO 700

603 MSG='DVODE-- ISTATE (=I1) .gt. 1 but DVODE not initialized '

CALL XERRWD (MSG, 60, 3, 1, 1, ISTATE, 0, 0, ZERO, ZERO)

GO TO 700

604 MSG = 'DVODE-- NEQ (=I1) .lt. 1 '

CALL XERRWD (MSG, 30, 4, 1, 1, NEQ, 0, 0, ZERO, ZERO)

GO TO 700

605 MSG = 'DVODE-- ISTATE = 3 and NEQ increased (I1 to I2) '

CALL XERRWD (MSG, 50, 5, 1, 2, N, NEQ, 0, ZERO, ZERO)

GO TO 700

606 MSG = 'DVODE-- ITOL (=I1) illegal '

CALL XERRWD (MSG, 30, 6, 1, 1, ITOL, 0, 0, ZERO, ZERO)

GO TO 700

607 MSG = 'DVODE-- IOPT (=I1) illegal '

CALL XERRWD (MSG, 30, 7, 1, 1, IOPT, 0, 0, ZERO, ZERO)

GO TO 700

608 MSG = 'DVODE-- MF (=I1) illegal '

CALL XERRWD (MSG, 30, 8, 1, 1, MF, 0, 0, ZERO, ZERO)

GO TO 700

609 MSG = 'DVODE-- ML (=I1) illegal.. .lt.0 or .ge.NEQ (=I2)'

CALL XERRWD (MSG, 50, 9, 1, 2, ML, NEQ, 0, ZERO, ZERO)

GO TO 700

610 MSG = 'DVODE-- MU (=I1) illegal.. .lt.0 or .ge.NEQ (=I2)'

CALL XERRWD (MSG, 50, 10, 1, 2, MU, NEQ, 0, ZERO, ZERO)

GO TO 700

611 MSG = 'DVODE-- MAXORD (=I1) .lt. 0 '

CALL XERRWD (MSG, 30, 11, 1, 1, MAXORD, 0, 0, ZERO, ZERO)

GO TO 700

612 MSG = 'DVODE-- MXSTEP (=I1) .lt. 0 '

CALL XERRWD (MSG, 30, 12, 1, 1, MXSTEP, 0, 0, ZERO, ZERO)

GO TO 700

613 MSG = 'DVODE-- MXHNIL (=I1) .lt. 0 '

CALL XERRWD (MSG, 30, 13, 1, 1, MXHNIL, 0, 0, ZERO, ZERO)

GO TO 700

614 MSG = 'DVODE-- TOUT (=R1) behind T (=R2) '

CALL XERRWD (MSG, 40, 14, 1, 0, 0, 0, 2, TOUT, T)

MSG = ' integration direction is given by H0 (=R1) '

CALL XERRWD (MSG, 50, 14, 1, 0, 0, 0, 1, H0, ZERO)

GO TO 700

615 MSG = 'DVODE-- HMAX (=R1) .lt. 0.0 '

CALL XERRWD (MSG, 30, 15, 1, 0, 0, 0, 1, HMAX, ZERO)

GO TO 700

616 MSG = 'DVODE-- HMIN (=R1) .lt. 0.0 '

CALL XERRWD (MSG, 30, 16, 1, 0, 0, 0, 1, HMIN, ZERO)

GO TO 700

617 CONTINUE

MSG='DVODE-- RWORK length needed, LENRW (=I1), exceeds LRW (=I2)'

CALL XERRWD (MSG, 60, 17, 1, 2, LENRW, LRW, 0, ZERO, ZERO)

GO TO 700

618 CONTINUE

MSG='DVODE-- IWORK length needed, LENIW (=I1), exceeds LIW (=I2)'

CALL XERRWD (MSG, 60, 18, 1, 2, LENIW, LIW, 0, ZERO, ZERO)

GO TO 700

619 MSG = 'DVODE-- RTOL(I1) is R1 .lt. 0.0 '

CALL XERRWD (MSG, 40, 19, 1, 1, I, 0, 1, RTOLI, ZERO)

GO TO 700

620 MSG = 'DVODE-- ATOL(I1) is R1 .lt. 0.0 '

CALL XERRWD (MSG, 40, 20, 1, 1, I, 0, 1, ATOLI, ZERO)

GO TO 700

621 EWTI = RWORK(LEWT+I-1)

MSG = 'DVODE-- EWT(I1) is R1 .le. 0.0 '

CALL XERRWD (MSG, 40, 21, 1, 1, I, 0, 1, EWTI, ZERO)

GO TO 700

622 CONTINUE

MSG='DVODE-- TOUT (=R1) too close to T(=R2) to start integration'

CALL XERRWD (MSG, 60, 22, 1, 0, 0, 0, 2, TOUT, T)

GO TO 700

623 CONTINUE

MSG='DVODE-- ITASK = I1 and TOUT (=R1) behind TCUR - HU (= R2) '

CALL XERRWD (MSG, 60, 23, 1, 1, ITASK, 0, 2, TOUT, TP)

GO TO 700

624 CONTINUE

MSG='DVODE-- ITASK = 4 or 5 and TCRIT (=R1) behind TCUR (=R2) '

CALL XERRWD (MSG, 60, 24, 1, 0, 0, 0, 2, TCRIT, TN)

GO TO 700

625 CONTINUE

MSG='DVODE-- ITASK = 4 or 5 and TCRIT (=R1) behind TOUT (=R2) '

CALL XERRWD (MSG, 60, 25, 1, 0, 0, 0, 2, TCRIT, TOUT)

GO TO 700

626 MSG = 'DVODE-- At start of problem, too much accuracy '

CALL XERRWD (MSG, 50, 26, 1, 0, 0, 0, 0, ZERO, ZERO)

MSG=' requested for precision of machine.. see TOLSF (=R1) '

CALL XERRWD (MSG, 60, 26, 1, 0, 0, 0, 1, TOLSF, ZERO)

RWORK(14) = TOLSF

GO TO 700

627 MSG='DVODE-- Trouble from DVINDY. ITASK = I1, TOUT = R1. '

CALL XERRWD (MSG, 60, 27, 1, 1, ITASK, 0, 1, TOUT, ZERO)

C

700 CONTINUE

ISTATE = -3

RETURN

C

800 MSG = 'DVODE-- Run aborted.. apparent infinite loop '

CALL XERRWD (MSG, 50, 303, 2, 0, 0, 0, 0, ZERO, ZERO)

RETURN

C----------------------- End of Subroutine DVODE -----------------------

END

\*DECK DVHIN

SUBROUTINE DVHIN (N, T0, Y0, YDOT, F, RPAR, IPAR, TOUT, UROUND,

1 EWT, ITOL, ATOL, Y, TEMP, H0, NITER, IER)

EXTERNAL F

DOUBLE PRECISION T0, Y0, YDOT, RPAR, TOUT, UROUND, EWT, ATOL, Y,

1 TEMP, H0

INTEGER N, IPAR, ITOL, NITER, IER

DIMENSION Y0(\*), YDOT(\*), EWT(\*), ATOL(\*), Y(\*),

1 TEMP(\*), RPAR(\*), IPAR(\*)

C-----------------------------------------------------------------------

C Call sequence input -- N, T0, Y0, YDOT, F, RPAR, IPAR, TOUT, UROUND,

C EWT, ITOL, ATOL, Y, TEMP

C Call sequence output -- H0, NITER, IER

C COMMON block variables accessed -- None

C

C Subroutines called by DVHIN.. F

C Function routines called by DVHIN.. DVNORM

C-----------------------------------------------------------------------

C This routine computes the step size, H0, to be attempted on the

C first step, when the user has not supplied a value for this.

C

C First we check that TOUT - T0 differs significantly from zero. Then

C an iteration is done to approximate the initial second derivative

C and this is used to define h from w.r.m.s.norm(h\*\*2 \* yddot / 2) = 1.

C A bias factor of 1/2 is applied to the resulting h.

C The sign of H0 is inferred from the initial values of TOUT and T0.

C

C Communication with DVHIN is done with the following variables..

C

C N = Size of ODE system, input.

C T0 = Initial value of independent variable, input.

C Y0 = Vector of initial conditions, input.

C YDOT = Vector of initial first derivatives, input.

C F = Name of subroutine for right-hand side f(t,y), input.

C RPAR, IPAR = Dummy names for user's real and integer work arrays.

C TOUT = First output value of independent variable

C UROUND = Machine unit roundoff

C EWT, ITOL, ATOL = Error weights and tolerance parameters

C as described in the driver routine, input.

C Y, TEMP = Work arrays of length N.

C H0 = Step size to be attempted, output.

C NITER = Number of iterations (and of f evaluations) to compute H0,

C output.

C IER = The error flag, returned with the value

C IER = 0 if no trouble occurred, or

C IER = -1 if TOUT and T0 are considered too close to proceed.

C-----------------------------------------------------------------------

C

C Type declarations for local variables --------------------------------

C

DOUBLE PRECISION AFI, ATOLI, DELYI, H, HALF, HG, HLB, HNEW, HRAT,

1 HUB, HUN, PT1, T1, TDIST, TROUND, TWO, YDDNRM

INTEGER I, ITER

C

C Type declaration for function subroutines called ---------------------

C

DOUBLE PRECISION DVNORM

C-----------------------------------------------------------------------

C The following Fortran-77 declaration is to cause the values of the

C listed (local) variables to be saved between calls to this integrator.

C-----------------------------------------------------------------------

SAVE HALF, HUN, PT1, TWO

DATA HALF /0.5D0/, HUN /100.0D0/, PT1 /0.1D0/, TWO /2.0D0/

C

NITER = 0

TDIST = ABS(TOUT - T0)

TROUND = UROUND\*MAX(ABS(T0),ABS(TOUT))

IF (TDIST .LT. TWO\*TROUND) GO TO 100

C

C Set a lower bound on h based on the roundoff level in T0 and TOUT. ---

HLB = HUN\*TROUND

C Set an upper bound on h based on TOUT-T0 and the initial Y and YDOT. -

HUB = PT1\*TDIST

ATOLI = ATOL(1)

DO 10 I = 1, N

IF (ITOL .EQ. 2 .OR. ITOL .EQ. 4) ATOLI = ATOL(I)

DELYI = PT1\*ABS(Y0(I)) + ATOLI

AFI = ABS(YDOT(I))

IF (AFI\*HUB .GT. DELYI) HUB = DELYI/AFI

10 CONTINUE

C

C Set initial guess for h as geometric mean of upper and lower bounds. -

ITER = 0

HG = SQRT(HLB\*HUB)

C If the bounds have crossed, exit with the mean value. ----------------

IF (HUB .LT. HLB) THEN

H0 = HG

GO TO 90

ENDIF

C

C Looping point for iteration. -----------------------------------------

50 CONTINUE

C Estimate the second derivative as a difference quotient in f. --------

H = SIGN (HG, TOUT - T0)

T1 = T0 + H

DO 60 I = 1, N

60 Y(I) = Y0(I) + H\*YDOT(I)

CALL F (N, T1, Y, TEMP, RPAR, IPAR)

DO 70 I = 1, N

70 TEMP(I) = (TEMP(I) - YDOT(I))/H

YDDNRM = DVNORM (N, TEMP, EWT)

C Get the corresponding new value of h. --------------------------------

IF (YDDNRM\*HUB\*HUB .GT. TWO) THEN

HNEW = SQRT(TWO/YDDNRM)

ELSE

HNEW = SQRT(HG\*HUB)

ENDIF

ITER = ITER + 1

C-----------------------------------------------------------------------

C Test the stopping conditions.

C Stop if the new and previous h values differ by a factor of .lt. 2.

C Stop if four iterations have been done. Also, stop with previous h

C if HNEW/HG .gt. 2 after first iteration, as this probably means that

C the second derivative value is bad because of cancellation error.

C-----------------------------------------------------------------------

IF (ITER .GE. 4) GO TO 80

HRAT = HNEW/HG

IF ( (HRAT .GT. HALF) .AND. (HRAT .LT. TWO) ) GO TO 80

IF ( (ITER .GE. 2) .AND. (HNEW .GT. TWO\*HG) ) THEN

HNEW = HG

GO TO 80

ENDIF

HG = HNEW

GO TO 50

C

C Iteration done. Apply bounds, bias factor, and sign. Then exit. ----

80 H0 = HNEW\*HALF

IF (H0 .LT. HLB) H0 = HLB

IF (H0 .GT. HUB) H0 = HUB

90 H0 = SIGN(H0, TOUT - T0)

NITER = ITER

IER = 0

RETURN

C Error return for TOUT - T0 too small. --------------------------------

100 IER = -1

RETURN

C----------------------- End of Subroutine DVHIN -----------------------

END

\*DECK DVINDY

SUBROUTINE DVINDY (T, K, YH, LDYH, DKY, IFLAG)

DOUBLE PRECISION T, YH, DKY

INTEGER K, LDYH, IFLAG

DIMENSION YH(LDYH,\*), DKY(\*)

C-----------------------------------------------------------------------

C Call sequence input -- T, K, YH, LDYH

C Call sequence output -- DKY, IFLAG

C COMMON block variables accessed..

C /DVOD01/ -- H, TN, UROUND, L, N, NQ

C /DVOD02/ -- HU

C

C Subroutines called by DVINDY.. DSCAL, XERRWD

C Function routines called by DVINDY.. None

C-----------------------------------------------------------------------

C DVINDY computes interpolated values of the K-th derivative of the

C dependent variable vector y, and stores it in DKY. This routine

C is called within the package with K = 0 and T = TOUT, but may

C also be called by the user for any K up to the current order.

C (See detailed instructions in the usage documentation.)

C-----------------------------------------------------------------------

C The computed values in DKY are gotten by interpolation using the

C Nordsieck history array YH. This array corresponds uniquely to a

C vector-valued polynomial of degree NQCUR or less, and DKY is set

C to the K-th derivative of this polynomial at T.

C The formula for DKY is..

C q

C DKY(i) = sum c(j,K) \* (T - TN)\*\*(j-K) \* H\*\*(-j) \* YH(i,j+1)

C j=K

C where c(j,K) = j\*(j-1)\*...\*(j-K+1), q = NQCUR, TN = TCUR, H = HCUR.

C The quantities NQ = NQCUR, L = NQ+1, N, TN, and H are

C communicated by COMMON. The above sum is done in reverse order.

C IFLAG is returned negative if either K or T is out of bounds.

C

C Discussion above and comments in driver explain all variables.

C-----------------------------------------------------------------------

C

C Type declarations for labeled COMMON block DVOD01 --------------------

C

DOUBLE PRECISION ACNRM, CCMXJ, CONP, CRATE, DRC, EL,

1 ETA, ETAMAX, H, HMIN, HMXI, HNEW, HSCAL, PRL1,

2 RC, RL1, TAU, TQ, TN, UROUND

INTEGER ICF, INIT, IPUP, JCUR, JSTART, JSV, KFLAG, KUTH,

1 L, LMAX, LYH, LEWT, LACOR, LSAVF, LWM, LIWM,

2 LOCJS, MAXORD, METH, MITER, MSBJ, MXHNIL, MXSTEP,

3 N, NEWH, NEWQ, NHNIL, NQ, NQNYH, NQWAIT, NSLJ,

4 NSLP, NYH

C

C Type declarations for labeled COMMON block DVOD02 --------------------

C

DOUBLE PRECISION HU

INTEGER NCFN, NETF, NFE, NJE, NLU, NNI, NQU, NST

C

C Type declarations for local variables --------------------------------

C

DOUBLE PRECISION C, HUN, R, S, TFUZZ, TN1, TP, ZERO

INTEGER I, IC, J, JB, JB2, JJ, JJ1, JP1

CHARACTER\*80 MSG

C-----------------------------------------------------------------------

C The following Fortran-77 declaration is to cause the values of the

C listed (local) variables to be saved between calls to this integrator.

C-----------------------------------------------------------------------

SAVE HUN, ZERO

C

COMMON /DVOD01/ ACNRM, CCMXJ, CONP, CRATE, DRC, EL(13),

1 ETA, ETAMAX, H, HMIN, HMXI, HNEW, HSCAL, PRL1,

2 RC, RL1, TAU(13), TQ(5), TN, UROUND,

3 ICF, INIT, IPUP, JCUR, JSTART, JSV, KFLAG, KUTH,

4 L, LMAX, LYH, LEWT, LACOR, LSAVF, LWM, LIWM,

5 LOCJS, MAXORD, METH, MITER, MSBJ, MXHNIL, MXSTEP,

6 N, NEWH, NEWQ, NHNIL, NQ, NQNYH, NQWAIT, NSLJ,

7 NSLP, NYH

COMMON /DVOD02/ HU, NCFN, NETF, NFE, NJE, NLU, NNI, NQU, NST

C

DATA HUN /100.0D0/, ZERO /0.0D0/

C

IFLAG = 0

IF (K .LT. 0 .OR. K .GT. NQ) GO TO 80

TFUZZ = HUN\*UROUND\*(TN + HU)

TP = TN - HU - TFUZZ

TN1 = TN + TFUZZ

IF ((T-TP)\*(T-TN1) .GT. ZERO) GO TO 90

C

S = (T - TN)/H

IC = 1

IF (K .EQ. 0) GO TO 15

JJ1 = L - K

DO 10 JJ = JJ1, NQ

10 IC = IC\*JJ

15 C = REAL(IC)

DO 20 I = 1, N

20 DKY(I) = C\*YH(I,L)

IF (K .EQ. NQ) GO TO 55

JB2 = NQ - K

DO 50 JB = 1, JB2

J = NQ - JB

JP1 = J + 1

IC = 1

IF (K .EQ. 0) GO TO 35

JJ1 = JP1 - K

DO 30 JJ = JJ1, J

30 IC = IC\*JJ

35 C = REAL(IC)

DO 40 I = 1, N

40 DKY(I) = C\*YH(I,JP1) + S\*DKY(I)

50 CONTINUE

IF (K .EQ. 0) RETURN

55 R = H\*\*(-K)

CALL DSCAL (N, R, DKY, 1)

RETURN

C

80 MSG = 'DVINDY-- K (=I1) illegal '

CALL XERRWD (MSG, 30, 51, 1, 1, K, 0, 0, ZERO, ZERO)

IFLAG = -1

RETURN

90 MSG = 'DVINDY-- T (=R1) illegal '

CALL XERRWD (MSG, 30, 52, 1, 0, 0, 0, 1, T, ZERO)

MSG=' T not in interval TCUR - HU (= R1) to TCUR (=R2) '

CALL XERRWD (MSG, 60, 52, 1, 0, 0, 0, 2, TP, TN)

IFLAG = -2

RETURN

C----------------------- End of Subroutine DVINDY ----------------------

END

\*DECK DVSTEP

SUBROUTINE DVSTEP (Y, YH, LDYH, YH1, EWT, SAVF, VSAV, ACOR,

1 WM, IWM, F, JAC, PSOL, VNLS, RPAR, IPAR)

EXTERNAL F, JAC, PSOL, VNLS

DOUBLE PRECISION Y, YH, YH1, EWT, SAVF, VSAV, ACOR, WM, RPAR

INTEGER LDYH, IWM, IPAR

DIMENSION Y(\*), YH(LDYH,\*), YH1(\*), EWT(\*), SAVF(\*), VSAV(\*),

1 ACOR(\*), WM(\*), IWM(\*), RPAR(\*), IPAR(\*)

C-----------------------------------------------------------------------

C Call sequence input -- Y, YH, LDYH, YH1, EWT, SAVF, VSAV,

C ACOR, WM, IWM, F, JAC, PSOL, VNLS, RPAR, IPAR

C Call sequence output -- YH, ACOR, WM, IWM

C COMMON block variables accessed..

C /DVOD01/ ACNRM, EL(13), H, HMIN, HMXI, HNEW, HSCAL, RC, TAU(13),

C TQ(5), TN, JCUR, JSTART, KFLAG, KUTH,

C L, LMAX, MAXORD, N, NEWQ, NQ, NQWAIT

C /DVOD02/ HU, NCFN, NETF, NFE, NQU, NST

C

C Subroutines called by DVSTEP.. F, DAXPY, DCOPY, DSCAL,

C DVJUST, VNLS, DVSET

C Function routines called by DVSTEP.. DVNORM

C-----------------------------------------------------------------------

C DVSTEP performs one step of the integration of an initial value

C problem for a system of ordinary differential equations.

C DVSTEP calls subroutine VNLS for the solution of the nonlinear system

C arising in the time step. Thus it is independent of the problem

C Jacobian structure and the type of nonlinear system solution method.

C DVSTEP returns a completion flag KFLAG (in COMMON).

C A return with KFLAG = -1 or -2 means either ABS(H) = HMIN or 10

C consecutive failures occurred. On a return with KFLAG negative,

C the values of TN and the YH array are as of the beginning of the last

C step, and H is the last step size attempted.

C

C Communication with DVSTEP is done with the following variables..

C

C Y = An array of length N used for the dependent variable vector.

C YH = An LDYH by LMAX array containing the dependent variables

C and their approximate scaled derivatives, where

C LMAX = MAXORD + 1. YH(i,j+1) contains the approximate

C j-th derivative of y(i), scaled by H\*\*j/factorial(j)

C (j = 0,1,...,NQ). On entry for the first step, the first

C two columns of YH must be set from the initial values.

C LDYH = A constant integer .ge. N, the first dimension of YH.

C N is the number of ODEs in the system.

C YH1 = A one-dimensional array occupying the same space as YH.

C EWT = An array of length N containing multiplicative weights

C for local error measurements. Local errors in y(i) are

C compared to 1.0/EWT(i) in various error tests.

C SAVF = An array of working storage, of length N.

C also used for input of YH(\*,MAXORD+2) when JSTART = -1

C and MAXORD .lt. the current order NQ.

C VSAV = A work array of length N passed to subroutine VNLS.

C ACOR = A work array of length N, used for the accumulated

C corrections. On a successful return, ACOR(i) contains

C the estimated one-step local error in y(i).

C WM,IWM = Real and integer work arrays associated with matrix

C operations in VNLS.

C F = Dummy name for the user supplied subroutine for f.

C JAC = Dummy name for the user supplied Jacobian subroutine.

C PSOL = Dummy name for the subroutine passed to VNLS, for

C possible use there.

C VNLS = Dummy name for the nonlinear system solving subroutine,

C whose real name is dependent on the method used.

C RPAR, IPAR = Dummy names for user's real and integer work arrays.

C-----------------------------------------------------------------------

C

C Type declarations for labeled COMMON block DVOD01 --------------------

C

DOUBLE PRECISION ACNRM, CCMXJ, CONP, CRATE, DRC, EL,

1 ETA, ETAMAX, H, HMIN, HMXI, HNEW, HSCAL, PRL1,

2 RC, RL1, TAU, TQ, TN, UROUND

INTEGER ICF, INIT, IPUP, JCUR, JSTART, JSV, KFLAG, KUTH,

1 L, LMAX, LYH, LEWT, LACOR, LSAVF, LWM, LIWM,

2 LOCJS, MAXORD, METH, MITER, MSBJ, MXHNIL, MXSTEP,

3 N, NEWH, NEWQ, NHNIL, NQ, NQNYH, NQWAIT, NSLJ,

4 NSLP, NYH

C

C Type declarations for labeled COMMON block DVOD02 --------------------

C

DOUBLE PRECISION HU

INTEGER NCFN, NETF, NFE, NJE, NLU, NNI, NQU, NST

C

C Type declarations for local variables --------------------------------

C

DOUBLE PRECISION ADDON, BIAS1,BIAS2,BIAS3, CNQUOT, DDN, DSM, DUP,

1 ETACF, ETAMIN, ETAMX1, ETAMX2, ETAMX3, ETAMXF,

2 ETAQ, ETAQM1, ETAQP1, FLOTL, ONE, ONEPSM,

3 R, THRESH, TOLD, ZERO

INTEGER I, I1, I2, IBACK, J, JB, KFC, KFH, MXNCF, NCF, NFLAG

C

C Type declaration for function subroutines called ---------------------

C

DOUBLE PRECISION DVNORM

C-----------------------------------------------------------------------

C The following Fortran-77 declaration is to cause the values of the

C listed (local) variables to be saved between calls to this integrator.

C-----------------------------------------------------------------------

SAVE ADDON, BIAS1, BIAS2, BIAS3,

1 ETACF, ETAMIN, ETAMX1, ETAMX2, ETAMX3, ETAMXF, ETAQ, ETAQM1,

2 KFC, KFH, MXNCF, ONEPSM, THRESH, ONE, ZERO

C-----------------------------------------------------------------------

COMMON /DVOD01/ ACNRM, CCMXJ, CONP, CRATE, DRC, EL(13),

1 ETA, ETAMAX, H, HMIN, HMXI, HNEW, HSCAL, PRL1,

2 RC, RL1, TAU(13), TQ(5), TN, UROUND,

3 ICF, INIT, IPUP, JCUR, JSTART, JSV, KFLAG, KUTH,

4 L, LMAX, LYH, LEWT, LACOR, LSAVF, LWM, LIWM,

5 LOCJS, MAXORD, METH, MITER, MSBJ, MXHNIL, MXSTEP,

6 N, NEWH, NEWQ, NHNIL, NQ, NQNYH, NQWAIT, NSLJ,

7 NSLP, NYH

COMMON /DVOD02/ HU, NCFN, NETF, NFE, NJE, NLU, NNI, NQU, NST

C

DATA KFC/-3/, KFH/-7/, MXNCF/10/

DATA ADDON /1.0D-6/, BIAS1 /6.0D0/, BIAS2 /6.0D0/,

1 BIAS3 /10.0D0/, ETACF /0.25D0/, ETAMIN /0.1D0/,

2 ETAMXF /0.2D0/, ETAMX1 /1.0D4/, ETAMX2 /10.0D0/,

3 ETAMX3 /10.0D0/, ONEPSM /1.00001D0/, THRESH /1.5D0/

DATA ONE/1.0D0/, ZERO/0.0D0/

C

KFLAG = 0

TOLD = TN

NCF = 0

JCUR = 0

NFLAG = 0

IF (JSTART .GT. 0) GO TO 20

IF (JSTART .EQ. -1) GO TO 100

C-----------------------------------------------------------------------

C On the first call, the order is set to 1, and other variables are

C initialized. ETAMAX is the maximum ratio by which H can be increased

C in a single step. It is normally 10, but is larger during the

C first step to compensate for the small initial H. If a failure

C occurs (in corrector convergence or error test), ETAMAX is set to 1

C for the next increase.

C-----------------------------------------------------------------------

LMAX = MAXORD + 1

NQ = 1

L = 2

NQNYH = NQ\*LDYH

TAU(1) = H

PRL1 = ONE

RC = ZERO

ETAMAX = ETAMX1

NQWAIT = 2

HSCAL = H

GO TO 200

C-----------------------------------------------------------------------

C Take preliminary actions on a normal continuation step (JSTART.GT.0).

C If the driver changed H, then ETA must be reset and NEWH set to 1.

C If a change of order was dictated on the previous step, then

C it is done here and appropriate adjustments in the history are made.

C On an order decrease, the history array is adjusted by DVJUST.

C On an order increase, the history array is augmented by a column.

C On a change of step size H, the history array YH is rescaled.

C-----------------------------------------------------------------------

20 CONTINUE

IF (KUTH .EQ. 1) THEN

ETA = MIN(ETA,H/HSCAL)

NEWH = 1

ENDIF

50 IF (NEWH .EQ. 0) GO TO 200

IF (NEWQ .EQ. NQ) GO TO 150

IF (NEWQ .LT. NQ) THEN

CALL DVJUST (YH, LDYH, -1)

NQ = NEWQ

L = NQ + 1

NQWAIT = L

GO TO 150

ENDIF

IF (NEWQ .GT. NQ) THEN

CALL DVJUST (YH, LDYH, 1)

NQ = NEWQ

L = NQ + 1

NQWAIT = L

GO TO 150

ENDIF

C-----------------------------------------------------------------------

C The following block handles preliminaries needed when JSTART = -1.

C If N was reduced, zero out part of YH to avoid undefined references.

C If MAXORD was reduced to a value less than the tentative order NEWQ,

C then NQ is set to MAXORD, and a new H ratio ETA is chosen.

C Otherwise, we take the same preliminary actions as for JSTART .gt. 0.

C In any case, NQWAIT is reset to L = NQ + 1 to prevent further

C changes in order for that many steps.

C The new H ratio ETA is limited by the input H if KUTH = 1,

C by HMIN if KUTH = 0, and by HMXI in any case.

C Finally, the history array YH is rescaled.

C-----------------------------------------------------------------------

100 CONTINUE

LMAX = MAXORD + 1

IF (N .EQ. LDYH) GO TO 120

I1 = 1 + (NEWQ + 1)\*LDYH

I2 = (MAXORD + 1)\*LDYH

IF (I1 .GT. I2) GO TO 120

DO 110 I = I1, I2

110 YH1(I) = ZERO

120 IF (NEWQ .LE. MAXORD) GO TO 140

FLOTL = REAL(LMAX)

IF (MAXORD .LT. NQ-1) THEN

DDN = DVNORM (N, SAVF, EWT)/TQ(1)

ETA = ONE/((BIAS1\*DDN)\*\*(ONE/FLOTL) + ADDON)

ENDIF

IF (MAXORD .EQ. NQ .AND. NEWQ .EQ. NQ+1) ETA = ETAQ

IF (MAXORD .EQ. NQ-1 .AND. NEWQ .EQ. NQ+1) THEN

ETA = ETAQM1

CALL DVJUST (YH, LDYH, -1)

ENDIF

IF (MAXORD .EQ. NQ-1 .AND. NEWQ .EQ. NQ) THEN

DDN = DVNORM (N, SAVF, EWT)/TQ(1)

ETA = ONE/((BIAS1\*DDN)\*\*(ONE/FLOTL) + ADDON)

CALL DVJUST (YH, LDYH, -1)

ENDIF

ETA = MIN(ETA,ONE)

NQ = MAXORD

L = LMAX

140 IF (KUTH .EQ. 1) ETA = MIN(ETA,ABS(H/HSCAL))

IF (KUTH .EQ. 0) ETA = MAX(ETA,HMIN/ABS(HSCAL))

ETA = ETA/MAX(ONE,ABS(HSCAL)\*HMXI\*ETA)

NEWH = 1

NQWAIT = L

IF (NEWQ .LE. MAXORD) GO TO 50

C Rescale the history array for a change in H by a factor of ETA. ------

150 R = ONE

DO 180 J = 2, L

R = R\*ETA

CALL DSCAL (N, R, YH(1,J), 1 )

180 CONTINUE

H = HSCAL\*ETA

HSCAL = H

RC = RC\*ETA

NQNYH = NQ\*LDYH

C-----------------------------------------------------------------------

C This section computes the predicted values by effectively

C multiplying the YH array by the Pascal triangle matrix.

C DVSET is called to calculate all integration coefficients.

C RC is the ratio of new to old values of the coefficient H/EL(2)=h/l1.

C-----------------------------------------------------------------------

200 TN = TN + H

I1 = NQNYH + 1

DO 220 JB = 1, NQ

I1 = I1 - LDYH

DO 210 I = I1, NQNYH

210 YH1(I) = YH1(I) + YH1(I+LDYH)

220 CONTINUE

CALL DVSET

RL1 = ONE/EL(2)

RC = RC\*(RL1/PRL1)

PRL1 = RL1

C

C Call the nonlinear system solver. ------------------------------------

C

CALL VNLS (Y, YH, LDYH, VSAV, SAVF, EWT, ACOR, IWM, WM,

1 F, JAC, PSOL, NFLAG, RPAR, IPAR)

C

IF (NFLAG .EQ. 0) GO TO 450

C-----------------------------------------------------------------------

C The VNLS routine failed to achieve convergence (NFLAG .NE. 0).

C The YH array is retracted to its values before prediction.

C The step size H is reduced and the step is retried, if possible.

C Otherwise, an error exit is taken.

C-----------------------------------------------------------------------

NCF = NCF + 1

NCFN = NCFN + 1

ETAMAX = ONE

TN = TOLD

I1 = NQNYH + 1

DO 430 JB = 1, NQ

I1 = I1 - LDYH

DO 420 I = I1, NQNYH

420 YH1(I) = YH1(I) - YH1(I+LDYH)

430 CONTINUE

IF (NFLAG .LT. -1) GO TO 680

IF (ABS(H) .LE. HMIN\*ONEPSM) GO TO 670

IF (NCF .EQ. MXNCF) GO TO 670

ETA = ETACF

ETA = MAX(ETA,HMIN/ABS(H))

NFLAG = -1

GO TO 150

C-----------------------------------------------------------------------

C The corrector has converged (NFLAG = 0). The local error test is

C made and control passes to statement 500 if it fails.

C-----------------------------------------------------------------------

450 CONTINUE

DSM = ACNRM/TQ(2)

IF (DSM .GT. ONE) GO TO 500

C-----------------------------------------------------------------------

C After a successful step, update the YH and TAU arrays and decrement

C NQWAIT. If NQWAIT is then 1 and NQ .lt. MAXORD, then ACOR is saved

C for use in a possible order increase on the next step.

C If ETAMAX = 1 (a failure occurred this step), keep NQWAIT .ge. 2.

C-----------------------------------------------------------------------

KFLAG = 0

NST = NST + 1

HU = H

NQU = NQ

DO 470 IBACK = 1, NQ

I = L - IBACK

470 TAU(I+1) = TAU(I)

TAU(1) = H

DO 480 J = 1, L

CALL DAXPY (N, EL(J), ACOR, 1, YH(1,J), 1 )

480 CONTINUE

NQWAIT = NQWAIT - 1

IF ((L .EQ. LMAX) .OR. (NQWAIT .NE. 1)) GO TO 490

CALL DCOPY (N, ACOR, 1, YH(1,LMAX), 1 )

CONP = TQ(5)

490 IF (ETAMAX .NE. ONE) GO TO 560

IF (NQWAIT .LT. 2) NQWAIT = 2

NEWQ = NQ

NEWH = 0

ETA = ONE

HNEW = H

GO TO 690

C-----------------------------------------------------------------------

C The error test failed. KFLAG keeps track of multiple failures.

C Restore TN and the YH array to their previous values, and prepare

C to try the step again. Compute the optimum step size for the

C same order. After repeated failures, H is forced to decrease

C more rapidly.

C-----------------------------------------------------------------------

500 KFLAG = KFLAG - 1

NETF = NETF + 1

NFLAG = -2

TN = TOLD

I1 = NQNYH + 1

DO 520 JB = 1, NQ

I1 = I1 - LDYH

DO 510 I = I1, NQNYH

510 YH1(I) = YH1(I) - YH1(I+LDYH)

520 CONTINUE

IF (ABS(H) .LE. HMIN\*ONEPSM) GO TO 660

ETAMAX = ONE

IF (KFLAG .LE. KFC) GO TO 530

C Compute ratio of new H to current H at the current order. ------------

FLOTL = REAL(L)

ETA = ONE/((BIAS2\*DSM)\*\*(ONE/FLOTL) + ADDON)

ETA = MAX(ETA,HMIN/ABS(H),ETAMIN)

IF ((KFLAG .LE. -2) .AND. (ETA .GT. ETAMXF)) ETA = ETAMXF

GO TO 150

C-----------------------------------------------------------------------

C Control reaches this section if 3 or more consecutive failures

C have occurred. It is assumed that the elements of the YH array

C have accumulated errors of the wrong order. The order is reduced

C by one, if possible. Then H is reduced by a factor of 0.1 and

C the step is retried. After a total of 7 consecutive failures,

C an exit is taken with KFLAG = -1.

C-----------------------------------------------------------------------

530 IF (KFLAG .EQ. KFH) GO TO 660

IF (NQ .EQ. 1) GO TO 540

ETA = MAX(ETAMIN,HMIN/ABS(H))

CALL DVJUST (YH, LDYH, -1)

L = NQ

NQ = NQ - 1

NQWAIT = L

GO TO 150

540 ETA = MAX(ETAMIN,HMIN/ABS(H))

H = H\*ETA

HSCAL = H

TAU(1) = H

CALL F (N, TN, Y, SAVF, RPAR, IPAR)

NFE = NFE + 1

DO 550 I = 1, N

550 YH(I,2) = H\*SAVF(I)

NQWAIT = 10

GO TO 200

C-----------------------------------------------------------------------

C If NQWAIT = 0, an increase or decrease in order by one is considered.

C Factors ETAQ, ETAQM1, ETAQP1 are computed by which H could

C be multiplied at order q, q-1, or q+1, respectively.

C The largest of these is determined, and the new order and

C step size set accordingly.

C A change of H or NQ is made only if H increases by at least a

C factor of THRESH. If an order change is considered and rejected,

C then NQWAIT is set to 2 (reconsider it after 2 steps).

C-----------------------------------------------------------------------

C Compute ratio of new H to current H at the current order. ------------

560 FLOTL = REAL(L)

ETAQ = ONE/((BIAS2\*DSM)\*\*(ONE/FLOTL) + ADDON)

IF (NQWAIT .NE. 0) GO TO 600

NQWAIT = 2

ETAQM1 = ZERO

IF (NQ .EQ. 1) GO TO 570

C Compute ratio of new H to current H at the current order less one. ---

DDN = DVNORM (N, YH(1,L), EWT)/TQ(1)

ETAQM1 = ONE/((BIAS1\*DDN)\*\*(ONE/(FLOTL - ONE)) + ADDON)

570 ETAQP1 = ZERO

IF (L .EQ. LMAX) GO TO 580

C Compute ratio of new H to current H at current order plus one. -------

CNQUOT = (TQ(5)/CONP)\*(H/TAU(2))\*\*L

DO 575 I = 1, N

575 SAVF(I) = ACOR(I) - CNQUOT\*YH(I,LMAX)

DUP = DVNORM (N, SAVF, EWT)/TQ(3)

ETAQP1 = ONE/((BIAS3\*DUP)\*\*(ONE/(FLOTL + ONE)) + ADDON)

580 IF (ETAQ .GE. ETAQP1) GO TO 590

IF (ETAQP1 .GT. ETAQM1) GO TO 620

GO TO 610

590 IF (ETAQ .LT. ETAQM1) GO TO 610

600 ETA = ETAQ

NEWQ = NQ

GO TO 630

610 ETA = ETAQM1

NEWQ = NQ - 1

GO TO 630

620 ETA = ETAQP1

NEWQ = NQ + 1

CALL DCOPY (N, ACOR, 1, YH(1,LMAX), 1)

C Test tentative new H against THRESH, ETAMAX, and HMXI, then exit. ----

630 IF (ETA .LT. THRESH .OR. ETAMAX .EQ. ONE) GO TO 640

ETA = MIN(ETA,ETAMAX)

ETA = ETA/MAX(ONE,ABS(H)\*HMXI\*ETA)

NEWH = 1

HNEW = H\*ETA

GO TO 690

640 NEWQ = NQ

NEWH = 0

ETA = ONE

HNEW = H

GO TO 690

C-----------------------------------------------------------------------

C All returns are made through this section.

C On a successful return, ETAMAX is reset and ACOR is scaled.

C-----------------------------------------------------------------------

660 KFLAG = -1

GO TO 720

670 KFLAG = -2

GO TO 720

680 IF (NFLAG .EQ. -2) KFLAG = -3

IF (NFLAG .EQ. -3) KFLAG = -4

GO TO 720

690 ETAMAX = ETAMX3

IF (NST .LE. 10) ETAMAX = ETAMX2

700 R = ONE/TQ(2)

CALL DSCAL (N, R, ACOR, 1)

720 JSTART = 1

RETURN

C----------------------- End of Subroutine DVSTEP ----------------------

END

\*DECK DVSET

SUBROUTINE DVSET

C-----------------------------------------------------------------------

C Call sequence communication.. None

C COMMON block variables accessed..

C /DVOD01/ -- EL(13), H, TAU(13), TQ(5), L(= NQ + 1),

C METH, NQ, NQWAIT

C

C Subroutines called by DVSET.. None

C Function routines called by DVSET.. None

C-----------------------------------------------------------------------

C DVSET is called by DVSTEP and sets coefficients for use there.

C

C For each order NQ, the coefficients in EL are calculated by use of

C the generating polynomial lambda(x), with coefficients EL(i).

C lambda(x) = EL(1) + EL(2)\*x + ... + EL(NQ+1)\*(x\*\*NQ).

C For the backward differentiation formulas,

C NQ-1

C lambda(x) = (1 + x/xi\*(NQ)) \* product (1 + x/xi(i) ) .

C i = 1

C For the Adams formulas,

C NQ-1

C (d/dx) lambda(x) = c \* product (1 + x/xi(i) ) ,

C i = 1

C lambda(-1) = 0, lambda(0) = 1,

C where c is a normalization constant.

C In both cases, xi(i) is defined by

C H\*xi(i) = t sub n - t sub (n-i)

C = H + TAU(1) + TAU(2) + ... TAU(i-1).

C

C

C In addition to variables described previously, communication

C with DVSET uses the following..

C TAU = A vector of length 13 containing the past NQ values

C of H.

C EL = A vector of length 13 in which vset stores the

C coefficients for the corrector formula.

C TQ = A vector of length 5 in which vset stores constants

C used for the convergence test, the error test, and the

C selection of H at a new order.

C METH = The basic method indicator.

C NQ = The current order.

C L = NQ + 1, the length of the vector stored in EL, and

C the number of columns of the YH array being used.

C NQWAIT = A counter controlling the frequency of order changes.

C An order change is about to be considered if NQWAIT = 1.

C-----------------------------------------------------------------------

C

C Type declarations for labeled COMMON block DVOD01 --------------------

C

DOUBLE PRECISION ACNRM, CCMXJ, CONP, CRATE, DRC, EL,

1 ETA, ETAMAX, H, HMIN, HMXI, HNEW, HSCAL, PRL1,

2 RC, RL1, TAU, TQ, TN, UROUND

INTEGER ICF, INIT, IPUP, JCUR, JSTART, JSV, KFLAG, KUTH,

1 L, LMAX, LYH, LEWT, LACOR, LSAVF, LWM, LIWM,

2 LOCJS, MAXORD, METH, MITER, MSBJ, MXHNIL, MXSTEP,

3 N, NEWH, NEWQ, NHNIL, NQ, NQNYH, NQWAIT, NSLJ,

4 NSLP, NYH

C

C Type declarations for local variables --------------------------------

C

DOUBLE PRECISION AHATN0, ALPH0, CNQM1, CORTES, CSUM, ELP, EM,

1 EM0, FLOTI, FLOTL, FLOTNQ, HSUM, ONE, RXI, RXIS, S, SIX,

2 T1, T2, T3, T4, T5, T6, TWO, XI, ZERO

INTEGER I, IBACK, J, JP1, NQM1, NQM2

C

DIMENSION EM(13)

C-----------------------------------------------------------------------

C The following Fortran-77 declaration is to cause the values of the

C listed (local) variables to be saved between calls to this integrator.

C-----------------------------------------------------------------------

SAVE CORTES, ONE, SIX, TWO, ZERO

C

COMMON /DVOD01/ ACNRM, CCMXJ, CONP, CRATE, DRC, EL(13),

1 ETA, ETAMAX, H, HMIN, HMXI, HNEW, HSCAL, PRL1,

2 RC, RL1, TAU(13), TQ(5), TN, UROUND,

3 ICF, INIT, IPUP, JCUR, JSTART, JSV, KFLAG, KUTH,

4 L, LMAX, LYH, LEWT, LACOR, LSAVF, LWM, LIWM,

5 LOCJS, MAXORD, METH, MITER, MSBJ, MXHNIL, MXSTEP,

6 N, NEWH, NEWQ, NHNIL, NQ, NQNYH, NQWAIT, NSLJ,

7 NSLP, NYH

C

DATA CORTES /0.1D0/

DATA ONE /1.0D0/, SIX /6.0D0/, TWO /2.0D0/, ZERO /0.0D0/

C

FLOTL = REAL(L)

NQM1 = NQ - 1

NQM2 = NQ - 2

GO TO (100, 200), METH

C

C Set coefficients for Adams methods. ----------------------------------

100 IF (NQ .NE. 1) GO TO 110

EL(1) = ONE

EL(2) = ONE

TQ(1) = ONE

TQ(2) = TWO

TQ(3) = SIX\*TQ(2)

TQ(5) = ONE

GO TO 300

110 HSUM = H

EM(1) = ONE

FLOTNQ = FLOTL - ONE

DO 115 I = 2, L

115 EM(I) = ZERO

DO 150 J = 1, NQM1

IF ((J .NE. NQM1) .OR. (NQWAIT .NE. 1)) GO TO 130

S = ONE

CSUM = ZERO

DO 120 I = 1, NQM1

CSUM = CSUM + S\*EM(I)/REAL(I+1)

120 S = -S

TQ(1) = EM(NQM1)/(FLOTNQ\*CSUM)

130 RXI = H/HSUM

DO 140 IBACK = 1, J

I = (J + 2) - IBACK

140 EM(I) = EM(I) + EM(I-1)\*RXI

HSUM = HSUM + TAU(J)

150 CONTINUE

C Compute integral from -1 to 0 of polynomial and of x times it. -------

S = ONE

EM0 = ZERO

CSUM = ZERO

DO 160 I = 1, NQ

FLOTI = REAL(I)

EM0 = EM0 + S\*EM(I)/FLOTI

CSUM = CSUM + S\*EM(I)/(FLOTI+ONE)

160 S = -S

C In EL, form coefficients of normalized integrated polynomial. --------

S = ONE/EM0

EL(1) = ONE

DO 170 I = 1, NQ

170 EL(I+1) = S\*EM(I)/REAL(I)

XI = HSUM/H

TQ(2) = XI\*EM0/CSUM

TQ(5) = XI/EL(L)

IF (NQWAIT .NE. 1) GO TO 300

C For higher order control constant, multiply polynomial by 1+x/xi(q). -

RXI = ONE/XI

DO 180 IBACK = 1, NQ

I = (L + 1) - IBACK

180 EM(I) = EM(I) + EM(I-1)\*RXI

C Compute integral of polynomial. --------------------------------------

S = ONE

CSUM = ZERO

DO 190 I = 1, L

CSUM = CSUM + S\*EM(I)/REAL(I+1)

190 S = -S

TQ(3) = FLOTL\*EM0/CSUM

GO TO 300

C

C Set coefficients for BDF methods. ------------------------------------

200 DO 210 I = 3, L

210 EL(I) = ZERO

EL(1) = ONE

EL(2) = ONE

ALPH0 = -ONE

AHATN0 = -ONE

HSUM = H

RXI = ONE

RXIS = ONE

IF (NQ .EQ. 1) GO TO 240

DO 230 J = 1, NQM2

C In EL, construct coefficients of (1+x/xi(1))\*...\*(1+x/xi(j+1)). ------

HSUM = HSUM + TAU(J)

RXI = H/HSUM

JP1 = J + 1

ALPH0 = ALPH0 - ONE/REAL(JP1)

DO 220 IBACK = 1, JP1

I = (J + 3) - IBACK

220 EL(I) = EL(I) + EL(I-1)\*RXI

230 CONTINUE

ALPH0 = ALPH0 - ONE/REAL(NQ)

RXIS = -EL(2) - ALPH0

HSUM = HSUM + TAU(NQM1)

RXI = H/HSUM

AHATN0 = -EL(2) - RXI

DO 235 IBACK = 1, NQ

I = (NQ + 2) - IBACK

235 EL(I) = EL(I) + EL(I-1)\*RXIS

240 T1 = ONE - AHATN0 + ALPH0

T2 = ONE + REAL(NQ)\*T1

TQ(2) = ABS(ALPH0\*T2/T1)

TQ(5) = ABS(T2/(EL(L)\*RXI/RXIS))

IF (NQWAIT .NE. 1) GO TO 300

CNQM1 = RXIS/EL(L)

T3 = ALPH0 + ONE/REAL(NQ)

T4 = AHATN0 + RXI

ELP = T3/(ONE - T4 + T3)

TQ(1) = ABS(ELP/CNQM1)

HSUM = HSUM + TAU(NQ)

RXI = H/HSUM

T5 = ALPH0 - ONE/REAL(NQ+1)

T6 = AHATN0 - RXI

ELP = T2/(ONE - T6 + T5)

TQ(3) = ABS(ELP\*RXI\*(FLOTL + ONE)\*T5)

300 TQ(4) = CORTES\*TQ(2)

RETURN

C----------------------- End of Subroutine DVSET -----------------------

END

\*DECK DVJUST

SUBROUTINE DVJUST (YH, LDYH, IORD)

DOUBLE PRECISION YH

INTEGER LDYH, IORD

DIMENSION YH(LDYH,\*)

C-----------------------------------------------------------------------

C Call sequence input -- YH, LDYH, IORD

C Call sequence output -- YH

C COMMON block input -- NQ, METH, LMAX, HSCAL, TAU(13), N

C COMMON block variables accessed..

C /DVOD01/ -- HSCAL, TAU(13), LMAX, METH, N, NQ,

C

C Subroutines called by DVJUST.. DAXPY

C Function routines called by DVJUST.. None

C-----------------------------------------------------------------------

C This subroutine adjusts the YH array on reduction of order,

C and also when the order is increased for the stiff option (METH = 2).

C Communication with DVJUST uses the following..

C IORD = An integer flag used when METH = 2 to indicate an order

C increase (IORD = +1) or an order decrease (IORD = -1).

C HSCAL = Step size H used in scaling of Nordsieck array YH.

C (If IORD = +1, DVJUST assumes that HSCAL = TAU(1).)

C See References 1 and 2 for details.

C-----------------------------------------------------------------------

C

C Type declarations for labeled COMMON block DVOD01 --------------------

C

DOUBLE PRECISION ACNRM, CCMXJ, CONP, CRATE, DRC, EL,

1 ETA, ETAMAX, H, HMIN, HMXI, HNEW, HSCAL, PRL1,

2 RC, RL1, TAU, TQ, TN, UROUND

INTEGER ICF, INIT, IPUP, JCUR, JSTART, JSV, KFLAG, KUTH,

1 L, LMAX, LYH, LEWT, LACOR, LSAVF, LWM, LIWM,

2 LOCJS, MAXORD, METH, MITER, MSBJ, MXHNIL, MXSTEP,

3 N, NEWH, NEWQ, NHNIL, NQ, NQNYH, NQWAIT, NSLJ,

4 NSLP, NYH

C

C Type declarations for local variables --------------------------------

C

DOUBLE PRECISION ALPH0, ALPH1, HSUM, ONE, PROD, T1, XI,XIOLD, ZERO

INTEGER I, IBACK, J, JP1, LP1, NQM1, NQM2, NQP1

C-----------------------------------------------------------------------

C The following Fortran-77 declaration is to cause the values of the

C listed (local) variables to be saved between calls to this integrator.

C-----------------------------------------------------------------------

SAVE ONE, ZERO

C

COMMON /DVOD01/ ACNRM, CCMXJ, CONP, CRATE, DRC, EL(13),

1 ETA, ETAMAX, H, HMIN, HMXI, HNEW, HSCAL, PRL1,

2 RC, RL1, TAU(13), TQ(5), TN, UROUND,

3 ICF, INIT, IPUP, JCUR, JSTART, JSV, KFLAG, KUTH,

4 L, LMAX, LYH, LEWT, LACOR, LSAVF, LWM, LIWM,

5 LOCJS, MAXORD, METH, MITER, MSBJ, MXHNIL, MXSTEP,

6 N, NEWH, NEWQ, NHNIL, NQ, NQNYH, NQWAIT, NSLJ,

7 NSLP, NYH

C

DATA ONE /1.0D0/, ZERO /0.0D0/

C

IF ((NQ .EQ. 2) .AND. (IORD .NE. 1)) RETURN

NQM1 = NQ - 1

NQM2 = NQ - 2

GO TO (100, 200), METH

C-----------------------------------------------------------------------

C Nonstiff option...

C Check to see if the order is being increased or decreased.

C-----------------------------------------------------------------------

100 CONTINUE

IF (IORD .EQ. 1) GO TO 180

C Order decrease. ------------------------------------------------------

DO 110 J = 1, LMAX

110 EL(J) = ZERO

EL(2) = ONE

HSUM = ZERO

DO 130 J = 1, NQM2

C Construct coefficients of x\*(x+xi(1))\*...\*(x+xi(j)). -----------------

HSUM = HSUM + TAU(J)

XI = HSUM/HSCAL

JP1 = J + 1

DO 120 IBACK = 1, JP1

I = (J + 3) - IBACK

120 EL(I) = EL(I)\*XI + EL(I-1)

130 CONTINUE

C Construct coefficients of integrated polynomial. ---------------------

DO 140 J = 2, NQM1

140 EL(J+1) = REAL(NQ)\*EL(J)/REAL(J)

C Subtract correction terms from YH array. -----------------------------

DO 170 J = 3, NQ

DO 160 I = 1, N

160 YH(I,J) = YH(I,J) - YH(I,L)\*EL(J)

170 CONTINUE

RETURN

C Order increase. ------------------------------------------------------

C Zero out next column in YH array. ------------------------------------

180 CONTINUE

LP1 = L + 1

DO 190 I = 1, N

190 YH(I,LP1) = ZERO

RETURN

C-----------------------------------------------------------------------

C Stiff option...

C Check to see if the order is being increased or decreased.

C-----------------------------------------------------------------------

200 CONTINUE

IF (IORD .EQ. 1) GO TO 300

C Order decrease. ------------------------------------------------------

DO 210 J = 1, LMAX

210 EL(J) = ZERO

EL(3) = ONE

HSUM = ZERO

DO 230 J = 1,NQM2

C Construct coefficients of x\*x\*(x+xi(1))\*...\*(x+xi(j)). ---------------

HSUM = HSUM + TAU(J)

XI = HSUM/HSCAL

JP1 = J + 1

DO 220 IBACK = 1, JP1

I = (J + 4) - IBACK

220 EL(I) = EL(I)\*XI + EL(I-1)

230 CONTINUE

C Subtract correction terms from YH array. -----------------------------

DO 250 J = 3,NQ

DO 240 I = 1, N

240 YH(I,J) = YH(I,J) - YH(I,L)\*EL(J)

250 CONTINUE

RETURN

C Order increase. ------------------------------------------------------

300 DO 310 J = 1, LMAX

310 EL(J) = ZERO

EL(3) = ONE

ALPH0 = -ONE

ALPH1 = ONE

PROD = ONE

XIOLD = ONE

HSUM = HSCAL

IF (NQ .EQ. 1) GO TO 340

DO 330 J = 1, NQM1

C Construct coefficients of x\*x\*(x+xi(1))\*...\*(x+xi(j)). ---------------

JP1 = J + 1

HSUM = HSUM + TAU(JP1)

XI = HSUM/HSCAL

PROD = PROD\*XI

ALPH0 = ALPH0 - ONE/REAL(JP1)

ALPH1 = ALPH1 + ONE/XI

DO 320 IBACK = 1, JP1

I = (J + 4) - IBACK

320 EL(I) = EL(I)\*XIOLD + EL(I-1)

XIOLD = XI

330 CONTINUE

340 CONTINUE

T1 = (-ALPH0 - ALPH1)/PROD

C Load column L + 1 in YH array. ---------------------------------------

LP1 = L + 1

DO 350 I = 1, N

350 YH(I,LP1) = T1\*YH(I,LMAX)

C Add correction terms to YH array. ------------------------------------

NQP1 = NQ + 1

DO 370 J = 3, NQP1

CALL DAXPY (N, EL(J), YH(1,LP1), 1, YH(1,J), 1 )

370 CONTINUE

RETURN

C----------------------- End of Subroutine DVJUST ----------------------

END

\*DECK DVNLSD

SUBROUTINE DVNLSD (Y, YH, LDYH, VSAV, SAVF, EWT, ACOR, IWM, WM,

1 F, JAC, PDUM, NFLAG, RPAR, IPAR)

EXTERNAL F, JAC, PDUM

DOUBLE PRECISION Y, YH, VSAV, SAVF, EWT, ACOR, WM, RPAR

INTEGER LDYH, IWM, NFLAG, IPAR

DIMENSION Y(\*), YH(LDYH,\*), VSAV(\*), SAVF(\*), EWT(\*), ACOR(\*),

1 IWM(\*), WM(\*), RPAR(\*), IPAR(\*)

C-----------------------------------------------------------------------

C Call sequence input -- Y, YH, LDYH, SAVF, EWT, ACOR, IWM, WM,

C F, JAC, NFLAG, RPAR, IPAR

C Call sequence output -- YH, ACOR, WM, IWM, NFLAG

C COMMON block variables accessed..

C /DVOD01/ ACNRM, CRATE, DRC, H, RC, RL1, TQ(5), TN, ICF,

C JCUR, METH, MITER, N, NSLP

C /DVOD02/ HU, NCFN, NETF, NFE, NJE, NLU, NNI, NQU, NST

C

C Subroutines called by DVNLSD.. F, DAXPY, DCOPY, DSCAL, DVJAC, DVSOL

C Function routines called by DVNLSD.. DVNORM

C-----------------------------------------------------------------------

C Subroutine DVNLSD is a nonlinear system solver, which uses functional

C iteration or a chord (modified Newton) method. For the chord method

C direct linear algebraic system solvers are used. Subroutine DVNLSD

C then handles the corrector phase of this integration package.

C

C Communication with DVNLSD is done with the following variables. (For

C more details, please see the comments in the driver subroutine.)

C

C Y = The dependent variable, a vector of length N, input.

C YH = The Nordsieck (Taylor) array, LDYH by LMAX, input

C and output. On input, it contains predicted values.

C LDYH = A constant .ge. N, the first dimension of YH, input.

C VSAV = Unused work array.

C SAVF = A work array of length N.

C EWT = An error weight vector of length N, input.

C ACOR = A work array of length N, used for the accumulated

C corrections to the predicted y vector.

C WM,IWM = Real and integer work arrays associated with matrix

C operations in chord iteration (MITER .ne. 0).

C F = Dummy name for user supplied routine for f.

C JAC = Dummy name for user supplied Jacobian routine.

C PDUM = Unused dummy subroutine name. Included for uniformity

C over collection of integrators.

C NFLAG = Input/output flag, with values and meanings as follows..

C INPUT

C 0 first call for this time step.

C -1 convergence failure in previous call to DVNLSD.

C -2 error test failure in DVSTEP.

C OUTPUT

C 0 successful completion of nonlinear solver.

C -1 convergence failure or singular matrix.

C -2 unrecoverable error in matrix preprocessing

C (cannot occur here).

C -3 unrecoverable error in solution (cannot occur

C here).

C RPAR, IPAR = Dummy names for user's real and integer work arrays.

C

C IPUP = Own variable flag with values and meanings as follows..

C 0, do not update the Newton matrix.

C MITER .ne. 0, update Newton matrix, because it is the

C initial step, order was changed, the error

C test failed, or an update is indicated by

C the scalar RC or step counter NST.

C

C For more details, see comments in driver subroutine.

C-----------------------------------------------------------------------

C Type declarations for labeled COMMON block DVOD01 --------------------

C

DOUBLE PRECISION ACNRM, CCMXJ, CONP, CRATE, DRC, EL,

1 ETA, ETAMAX, H, HMIN, HMXI, HNEW, HSCAL, PRL1,

2 RC, RL1, TAU, TQ, TN, UROUND

INTEGER ICF, INIT, IPUP, JCUR, JSTART, JSV, KFLAG, KUTH,

1 L, LMAX, LYH, LEWT, LACOR, LSAVF, LWM, LIWM,

2 LOCJS, MAXORD, METH, MITER, MSBJ, MXHNIL, MXSTEP,

3 N, NEWH, NEWQ, NHNIL, NQ, NQNYH, NQWAIT, NSLJ,

4 NSLP, NYH

C

C Type declarations for labeled COMMON block DVOD02 --------------------

C

DOUBLE PRECISION HU

INTEGER NCFN, NETF, NFE, NJE, NLU, NNI, NQU, NST

C

C Type declarations for local variables --------------------------------

C

DOUBLE PRECISION CCMAX, CRDOWN, CSCALE, DCON, DEL, DELP, ONE,

1 RDIV, TWO, ZERO

INTEGER I, IERPJ, IERSL, M, MAXCOR, MSBP

C

C Type declaration for function subroutines called ---------------------

C

DOUBLE PRECISION DVNORM

C-----------------------------------------------------------------------

C The following Fortran-77 declaration is to cause the values of the

C listed (local) variables to be saved between calls to this integrator.

C-----------------------------------------------------------------------

SAVE CCMAX, CRDOWN, MAXCOR, MSBP, RDIV, ONE, TWO, ZERO

C

COMMON /DVOD01/ ACNRM, CCMXJ, CONP, CRATE, DRC, EL(13),

1 ETA, ETAMAX, H, HMIN, HMXI, HNEW, HSCAL, PRL1,

2 RC, RL1, TAU(13), TQ(5), TN, UROUND,

3 ICF, INIT, IPUP, JCUR, JSTART, JSV, KFLAG, KUTH,

4 L, LMAX, LYH, LEWT, LACOR, LSAVF, LWM, LIWM,

5 LOCJS, MAXORD, METH, MITER, MSBJ, MXHNIL, MXSTEP,

6 N, NEWH, NEWQ, NHNIL, NQ, NQNYH, NQWAIT, NSLJ,

7 NSLP, NYH

COMMON /DVOD02/ HU, NCFN, NETF, NFE, NJE, NLU, NNI, NQU, NST

C

DATA CCMAX /0.3D0/, CRDOWN /0.3D0/, MAXCOR /3/, MSBP /20/,

1 RDIV /2.0D0/

DATA ONE /1.0D0/, TWO /2.0D0/, ZERO /0.0D0/

C-----------------------------------------------------------------------

C On the first step, on a change of method order, or after a

C nonlinear convergence failure with NFLAG = -2, set IPUP = MITER

C to force a Jacobian update when MITER .ne. 0.

C-----------------------------------------------------------------------

IF (JSTART .EQ. 0) NSLP = 0

IF (NFLAG .EQ. 0) ICF = 0

IF (NFLAG .EQ. -2) IPUP = MITER

IF ( (JSTART .EQ. 0) .OR. (JSTART .EQ. -1) ) IPUP = MITER

C If this is functional iteration, set CRATE .eq. 1 and drop to 220

IF (MITER .EQ. 0) THEN

CRATE = ONE

GO TO 220

ENDIF

C-----------------------------------------------------------------------

C RC is the ratio of new to old values of the coefficient H/EL(2)=h/l1.

C When RC differs from 1 by more than CCMAX, IPUP is set to MITER

C to force DVJAC to be called, if a Jacobian is involved.

C In any case, DVJAC is called at least every MSBP steps.

C-----------------------------------------------------------------------

DRC = ABS(RC-ONE)

IF (DRC .GT. CCMAX .OR. NST .GE. NSLP+MSBP) IPUP = MITER

C-----------------------------------------------------------------------

C Up to MAXCOR corrector iterations are taken. A convergence test is

C made on the r.m.s. norm of each correction, weighted by the error

C weight vector EWT. The sum of the corrections is accumulated in the

C vector ACOR(i). The YH array is not altered in the corrector loop.

C-----------------------------------------------------------------------

220 M = 0

DELP = ZERO

CALL DCOPY (N, YH(1,1), 1, Y, 1 )

CALL F (N, TN, Y, SAVF, RPAR, IPAR)

NFE = NFE + 1

IF (IPUP .LE. 0) GO TO 250

C-----------------------------------------------------------------------

C If indicated, the matrix P = I - h\*rl1\*J is reevaluated and

C preprocessed before starting the corrector iteration. IPUP is set

C to 0 as an indicator that this has been done.

C-----------------------------------------------------------------------

CALL DVJAC (Y, YH, LDYH, EWT, ACOR, SAVF, WM, IWM, F, JAC, IERPJ,

1 RPAR, IPAR)

IPUP = 0

RC = ONE

DRC = ZERO

CRATE = ONE

NSLP = NST

C If matrix is singular, take error return to force cut in step size. --

IF (IERPJ .NE. 0) GO TO 430

250 DO 260 I = 1,N

260 ACOR(I) = ZERO

C This is a looping point for the corrector iteration. -----------------

270 IF (MITER .NE. 0) GO TO 350

C-----------------------------------------------------------------------

C In the case of functional iteration, update Y directly from

C the result of the last function evaluation.

C-----------------------------------------------------------------------

DO 280 I = 1,N

280 SAVF(I) = RL1\*(H\*SAVF(I) - YH(I,2))

DO 290 I = 1,N

290 Y(I) = SAVF(I) - ACOR(I)

DEL = DVNORM (N, Y, EWT)

DO 300 I = 1,N

300 Y(I) = YH(I,1) + SAVF(I)

CALL DCOPY (N, SAVF, 1, ACOR, 1)

GO TO 400

C-----------------------------------------------------------------------

C In the case of the chord method, compute the corrector error,

C and solve the linear system with that as right-hand side and

C P as coefficient matrix. The correction is scaled by the factor

C 2/(1+RC) to account for changes in h\*rl1 since the last DVJAC call.

C-----------------------------------------------------------------------

350 DO 360 I = 1,N

360 Y(I) = (RL1\*H)\*SAVF(I) - (RL1\*YH(I,2) + ACOR(I))

CALL DVSOL (WM, IWM, Y, IERSL)

NNI = NNI + 1

IF (IERSL .GT. 0) GO TO 410

IF (METH .EQ. 2 .AND. RC .NE. ONE) THEN

CSCALE = TWO/(ONE + RC)

CALL DSCAL (N, CSCALE, Y, 1)

ENDIF

DEL = DVNORM (N, Y, EWT)

CALL DAXPY (N, ONE, Y, 1, ACOR, 1)

DO 380 I = 1,N

380 Y(I) = YH(I,1) + ACOR(I)

C-----------------------------------------------------------------------

C Test for convergence. If M .gt. 0, an estimate of the convergence

C rate constant is stored in CRATE, and this is used in the test.

C-----------------------------------------------------------------------

400 IF (M .NE. 0) CRATE = MAX(CRDOWN\*CRATE,DEL/DELP)

DCON = DEL\*MIN(ONE,CRATE)/TQ(4)

IF (DCON .LE. ONE) GO TO 450

M = M + 1

IF (M .EQ. MAXCOR) GO TO 410

IF (M .GE. 2 .AND. DEL .GT. RDIV\*DELP) GO TO 410

DELP = DEL

CALL F (N, TN, Y, SAVF, RPAR, IPAR)

NFE = NFE + 1

GO TO 270

C

410 IF (MITER .EQ. 0 .OR. JCUR .EQ. 1) GO TO 430

ICF = 1

IPUP = MITER

GO TO 220

C

430 CONTINUE

NFLAG = -1

ICF = 2

IPUP = MITER

RETURN

C

C Return for successful step. ------------------------------------------

450 NFLAG = 0

JCUR = 0

ICF = 0

IF (M .EQ. 0) ACNRM = DEL

IF (M .GT. 0) ACNRM = DVNORM (N, ACOR, EWT)

RETURN

C----------------------- End of Subroutine DVNLSD ----------------------

END

\*DECK DVJAC

SUBROUTINE DVJAC (Y, YH, LDYH, EWT, FTEM, SAVF, WM, IWM, F, JAC,

1 IERPJ, RPAR, IPAR)

EXTERNAL F, JAC

DOUBLE PRECISION Y, YH, EWT, FTEM, SAVF, WM, RPAR

INTEGER LDYH, IWM, IERPJ, IPAR

DIMENSION Y(\*), YH(LDYH,\*), EWT(\*), FTEM(\*), SAVF(\*),

1 WM(\*), IWM(\*), RPAR(\*), IPAR(\*)

C-----------------------------------------------------------------------

C Call sequence input -- Y, YH, LDYH, EWT, FTEM, SAVF, WM, IWM,

C F, JAC, RPAR, IPAR

C Call sequence output -- WM, IWM, IERPJ

C COMMON block variables accessed..

C /DVOD01/ CCMXJ, DRC, H, RL1, TN, UROUND, ICF, JCUR, LOCJS,

C MITER, MSBJ, N, NSLJ

C /DVOD02/ NFE, NST, NJE, NLU

C

C Subroutines called by DVJAC.. F, JAC, DACOPY, DCOPY, DGBFA, DGEFA,

C DSCAL

C Function routines called by DVJAC.. DVNORM

C-----------------------------------------------------------------------

C DVJAC is called by DVNLSD to compute and process the matrix

C P = I - h\*rl1\*J , where J is an approximation to the Jacobian.

C Here J is computed by the user-supplied routine JAC if

C MITER = 1 or 4, or by finite differencing if MITER = 2, 3, or 5.

C If MITER = 3, a diagonal approximation to J is used.

C If JSV = -1, J is computed from scratch in all cases.

C If JSV = 1 and MITER = 1, 2, 4, or 5, and if the saved value of J is

C considered acceptable, then P is constructed from the saved J.

C J is stored in wm and replaced by P. If MITER .ne. 3, P is then

C subjected to LU decomposition in preparation for later solution

C of linear systems with P as coefficient matrix. This is done

C by DGEFA if MITER = 1 or 2, and by DGBFA if MITER = 4 or 5.

C

C Communication with DVJAC is done with the following variables. (For

C more details, please see the comments in the driver subroutine.)

C Y = Vector containing predicted values on entry.

C YH = The Nordsieck array, an LDYH by LMAX array, input.

C LDYH = A constant .ge. N, the first dimension of YH, input.

C EWT = An error weight vector of length N.

C SAVF = Array containing f evaluated at predicted y, input.

C WM = Real work space for matrices. In the output, it containS

C the inverse diagonal matrix if MITER = 3 and the LU

C decomposition of P if MITER is 1, 2 , 4, or 5.

C Storage of matrix elements starts at WM(3).

C Storage of the saved Jacobian starts at WM(LOCJS).

C WM also contains the following matrix-related data..

C WM(1) = SQRT(UROUND), used in numerical Jacobian step.

C WM(2) = H\*RL1, saved for later use if MITER = 3.

C IWM = Integer work space containing pivot information,

C starting at IWM(31), if MITER is 1, 2, 4, or 5.

C IWM also contains band parameters ML = IWM(1) and

C MU = IWM(2) if MITER is 4 or 5.

C F = Dummy name for the user supplied subroutine for f.

C JAC = Dummy name for the user supplied Jacobian subroutine.

C RPAR, IPAR = Dummy names for user's real and integer work arrays.

C RL1 = 1/EL(2) (input).

C IERPJ = Output error flag, = 0 if no trouble, 1 if the P

C matrix is found to be singular.

C JCUR = Output flag to indicate whether the Jacobian matrix

C (or approximation) is now current.

C JCUR = 0 means J is not current.

C JCUR = 1 means J is current.

C-----------------------------------------------------------------------

C

C Type declarations for labeled COMMON block DVOD01 --------------------

C

DOUBLE PRECISION ACNRM, CCMXJ, CONP, CRATE, DRC, EL,

1 ETA, ETAMAX, H, HMIN, HMXI, HNEW, HSCAL, PRL1,

2 RC, RL1, TAU, TQ, TN, UROUND

INTEGER ICF, INIT, IPUP, JCUR, JSTART, JSV, KFLAG, KUTH,

1 L, LMAX, LYH, LEWT, LACOR, LSAVF, LWM, LIWM,

2 LOCJS, MAXORD, METH, MITER, MSBJ, MXHNIL, MXSTEP,

3 N, NEWH, NEWQ, NHNIL, NQ, NQNYH, NQWAIT, NSLJ,

4 NSLP, NYH

C

C Type declarations for labeled COMMON block DVOD02 --------------------

C

DOUBLE PRECISION HU

INTEGER NCFN, NETF, NFE, NJE, NLU, NNI, NQU, NST

C

C Type declarations for local variables --------------------------------

C

DOUBLE PRECISION CON, DI, FAC, HRL1, ONE, PT1, R, R0, SRUR, THOU,

1 YI, YJ, YJJ, ZERO

INTEGER I, I1, I2, IER, II, J, J1, JJ, JOK, LENP, MBA, MBAND,

1 MEB1, MEBAND, ML, ML3, MU, NP1

C

C Type declaration for function subroutines called ---------------------

C

DOUBLE PRECISION DVNORM

C-----------------------------------------------------------------------

C The following Fortran-77 declaration is to cause the values of the

C listed (local) variables to be saved between calls to this subroutine.

C-----------------------------------------------------------------------

SAVE ONE, PT1, THOU, ZERO

C-----------------------------------------------------------------------

COMMON /DVOD01/ ACNRM, CCMXJ, CONP, CRATE, DRC, EL(13),

1 ETA, ETAMAX, H, HMIN, HMXI, HNEW, HSCAL, PRL1,

2 RC, RL1, TAU(13), TQ(5), TN, UROUND,

3 ICF, INIT, IPUP, JCUR, JSTART, JSV, KFLAG, KUTH,

4 L, LMAX, LYH, LEWT, LACOR, LSAVF, LWM, LIWM,

5 LOCJS, MAXORD, METH, MITER, MSBJ, MXHNIL, MXSTEP,

6 N, NEWH, NEWQ, NHNIL, NQ, NQNYH, NQWAIT, NSLJ,

7 NSLP, NYH

COMMON /DVOD02/ HU, NCFN, NETF, NFE, NJE, NLU, NNI, NQU, NST

C

DATA ONE /1.0D0/, THOU /1000.0D0/, ZERO /0.0D0/, PT1 /0.1D0/

C

IERPJ = 0

HRL1 = H\*RL1

C See whether J should be evaluated (JOK = -1) or not (JOK = 1). -------

JOK = JSV

IF (JSV .EQ. 1) THEN

IF (NST .EQ. 0 .OR. NST .GT. NSLJ+MSBJ) JOK = -1

IF (ICF .EQ. 1 .AND. DRC .LT. CCMXJ) JOK = -1

IF (ICF .EQ. 2) JOK = -1

ENDIF

C End of setting JOK. --------------------------------------------------

C

IF (JOK .EQ. -1 .AND. MITER .EQ. 1) THEN

C If JOK = -1 and MITER = 1, call JAC to evaluate Jacobian. ------------

NJE = NJE + 1

NSLJ = NST

JCUR = 1

LENP = N\*N

DO 110 I = 1,LENP

110 WM(I+2) = ZERO

CALL JAC (N, TN, Y, 0, 0, WM(3), N, RPAR, IPAR)

IF (JSV .EQ. 1) CALL DCOPY (LENP, WM(3), 1, WM(LOCJS), 1)

ENDIF

C

IF (JOK .EQ. -1 .AND. MITER .EQ. 2) THEN

C If MITER = 2, make N calls to F to approximate the Jacobian. ---------

NJE = NJE + 1

NSLJ = NST

JCUR = 1

FAC = DVNORM (N, SAVF, EWT)

R0 = THOU\*ABS(H)\*UROUND\*REAL(N)\*FAC

IF (R0 .EQ. ZERO) R0 = ONE

SRUR = WM(1)

J1 = 2

DO 230 J = 1,N

YJ = Y(J)

R = MAX(SRUR\*ABS(YJ),R0/EWT(J))

Y(J) = Y(J) + R

FAC = ONE/R

CALL F (N, TN, Y, FTEM, RPAR, IPAR)

DO 220 I = 1,N

220 WM(I+J1) = (FTEM(I) - SAVF(I))\*FAC

Y(J) = YJ

J1 = J1 + N

230 CONTINUE

NFE = NFE + N

LENP = N\*N

IF (JSV .EQ. 1) CALL DCOPY (LENP, WM(3), 1, WM(LOCJS), 1)

ENDIF

C

IF (JOK .EQ. 1 .AND. (MITER .EQ. 1 .OR. MITER .EQ. 2)) THEN

JCUR = 0

LENP = N\*N

CALL DCOPY (LENP, WM(LOCJS), 1, WM(3), 1)

ENDIF

C

IF (MITER .EQ. 1 .OR. MITER .EQ. 2) THEN

C Multiply Jacobian by scalar, add identity, and do LU decomposition. --

CON = -HRL1

CALL DSCAL (LENP, CON, WM(3), 1)

J = 3

NP1 = N + 1

DO 250 I = 1,N

WM(J) = WM(J) + ONE

250 J = J + NP1

NLU = NLU + 1

CALL DGEFA (WM(3), N, N, IWM(31), IER)

IF (IER .NE. 0) IERPJ = 1

RETURN

ENDIF

C End of code block for MITER = 1 or 2. --------------------------------

C

IF (MITER .EQ. 3) THEN

C If MITER = 3, construct a diagonal approximation to J and P. ---------

NJE = NJE + 1

JCUR = 1

WM(2) = HRL1

R = RL1\*PT1

DO 310 I = 1,N

310 Y(I) = Y(I) + R\*(H\*SAVF(I) - YH(I,2))

CALL F (N, TN, Y, WM(3), RPAR, IPAR)

NFE = NFE + 1

DO 320 I = 1,N

R0 = H\*SAVF(I) - YH(I,2)

DI = PT1\*R0 - H\*(WM(I+2) - SAVF(I))

WM(I+2) = ONE

IF (ABS(R0) .LT. UROUND/EWT(I)) GO TO 320

IF (ABS(DI) .EQ. ZERO) GO TO 330

WM(I+2) = PT1\*R0/DI

320 CONTINUE

RETURN

330 IERPJ = 1

RETURN

ENDIF

C End of code block for MITER = 3. -------------------------------------

C

C Set constants for MITER = 4 or 5. ------------------------------------

ML = IWM(1)

MU = IWM(2)

ML3 = ML + 3

MBAND = ML + MU + 1

MEBAND = MBAND + ML

LENP = MEBAND\*N

C

IF (JOK .EQ. -1 .AND. MITER .EQ. 4) THEN

C If JOK = -1 and MITER = 4, call JAC to evaluate Jacobian. ------------

NJE = NJE + 1

NSLJ = NST

JCUR = 1

DO 410 I = 1,LENP

410 WM(I+2) = ZERO

CALL JAC (N, TN, Y, ML, MU, WM(ML3), MEBAND, RPAR, IPAR)

IF (JSV .EQ. 1)

1 CALL DACOPY (MBAND, N, WM(ML3), MEBAND, WM(LOCJS), MBAND)

ENDIF

C

IF (JOK .EQ. -1 .AND. MITER .EQ. 5) THEN

C If MITER = 5, make ML+MU+1 calls to F to approximate the Jacobian. ---

NJE = NJE + 1

NSLJ = NST

JCUR = 1

MBA = MIN(MBAND,N)

MEB1 = MEBAND - 1

SRUR = WM(1)

FAC = DVNORM (N, SAVF, EWT)

R0 = THOU\*ABS(H)\*UROUND\*REAL(N)\*FAC

IF (R0 .EQ. ZERO) R0 = ONE

DO 560 J = 1,MBA

DO 530 I = J,N,MBAND

YI = Y(I)

R = MAX(SRUR\*ABS(YI),R0/EWT(I))

530 Y(I) = Y(I) + R

CALL F (N, TN, Y, FTEM, RPAR, IPAR)

DO 550 JJ = J,N,MBAND

Y(JJ) = YH(JJ,1)

YJJ = Y(JJ)

R = MAX(SRUR\*ABS(YJJ),R0/EWT(JJ))

FAC = ONE/R

I1 = MAX(JJ-MU,1)

I2 = MIN(JJ+ML,N)

II = JJ\*MEB1 - ML + 2

DO 540 I = I1,I2

540 WM(II+I) = (FTEM(I) - SAVF(I))\*FAC

550 CONTINUE

560 CONTINUE

NFE = NFE + MBA

IF (JSV .EQ. 1)

1 CALL DACOPY (MBAND, N, WM(ML3), MEBAND, WM(LOCJS), MBAND)

ENDIF

C

IF (JOK .EQ. 1) THEN

JCUR = 0

CALL DACOPY (MBAND, N, WM(LOCJS), MBAND, WM(ML3), MEBAND)

ENDIF

C

C Multiply Jacobian by scalar, add identity, and do LU decomposition.

CON = -HRL1

CALL DSCAL (LENP, CON, WM(3), 1 )

II = MBAND + 2

DO 580 I = 1,N

WM(II) = WM(II) + ONE

580 II = II + MEBAND

NLU = NLU + 1

CALL DGBFA (WM(3), MEBAND, N, ML, MU, IWM(31), IER)

IF (IER .NE. 0) IERPJ = 1

RETURN

C End of code block for MITER = 4 or 5. --------------------------------

C

C----------------------- End of Subroutine DVJAC -----------------------

END

\*DECK DACOPY

SUBROUTINE DACOPY (NROW, NCOL, A, NROWA, B, NROWB)

DOUBLE PRECISION A, B

INTEGER NROW, NCOL, NROWA, NROWB

DIMENSION A(NROWA,NCOL), B(NROWB,NCOL)

C-----------------------------------------------------------------------

C Call sequence input -- NROW, NCOL, A, NROWA, NROWB

C Call sequence output -- B

C COMMON block variables accessed -- None

C

C Subroutines called by DACOPY.. DCOPY

C Function routines called by DACOPY.. None

C-----------------------------------------------------------------------

C This routine copies one rectangular array, A, to another, B,

C where A and B may have different row dimensions, NROWA and NROWB.

C The data copied consists of NROW rows and NCOL columns.

C-----------------------------------------------------------------------

INTEGER IC

C

DO 20 IC = 1,NCOL

CALL DCOPY (NROW, A(1,IC), 1, B(1,IC), 1)

20 CONTINUE

C

RETURN

C----------------------- End of Subroutine DACOPY ----------------------

END

\*DECK DVSOL

SUBROUTINE DVSOL (WM, IWM, X, IERSL)

DOUBLE PRECISION WM, X

INTEGER IWM, IERSL

DIMENSION WM(\*), IWM(\*), X(\*)

C-----------------------------------------------------------------------

C Call sequence input -- WM, IWM, X

C Call sequence output -- X, IERSL

C COMMON block variables accessed..

C /DVOD01/ -- H, RL1, MITER, N

C

C Subroutines called by DVSOL.. DGESL, DGBSL

C Function routines called by DVSOL.. None

C-----------------------------------------------------------------------

C This routine manages the solution of the linear system arising from

C a chord iteration. It is called if MITER .ne. 0.

C If MITER is 1 or 2, it calls DGESL to accomplish this.

C If MITER = 3 it updates the coefficient H\*RL1 in the diagonal

C matrix, and then computes the solution.

C If MITER is 4 or 5, it calls DGBSL.

C Communication with DVSOL uses the following variables..

C WM = Real work space containing the inverse diagonal matrix if

C MITER = 3 and the LU decomposition of the matrix otherwise.

C Storage of matrix elements starts at WM(3).

C WM also contains the following matrix-related data..

C WM(1) = SQRT(UROUND) (not used here),

C WM(2) = HRL1, the previous value of H\*RL1, used if MITER = 3.

C IWM = Integer work space containing pivot information, starting at

C IWM(31), if MITER is 1, 2, 4, or 5. IWM also contains band

C parameters ML = IWM(1) and MU = IWM(2) if MITER is 4 or 5.

C X = The right-hand side vector on input, and the solution vector

C on output, of length N.

C IERSL = Output flag. IERSL = 0 if no trouble occurred.

C IERSL = 1 if a singular matrix arose with MITER = 3.

C-----------------------------------------------------------------------

C

C Type declarations for labeled COMMON block DVOD01 --------------------

C

DOUBLE PRECISION ACNRM, CCMXJ, CONP, CRATE, DRC, EL,

1 ETA, ETAMAX, H, HMIN, HMXI, HNEW, HSCAL, PRL1,

2 RC, RL1, TAU, TQ, TN, UROUND

INTEGER ICF, INIT, IPUP, JCUR, JSTART, JSV, KFLAG, KUTH,

1 L, LMAX, LYH, LEWT, LACOR, LSAVF, LWM, LIWM,

2 LOCJS, MAXORD, METH, MITER, MSBJ, MXHNIL, MXSTEP,

3 N, NEWH, NEWQ, NHNIL, NQ, NQNYH, NQWAIT, NSLJ,

4 NSLP, NYH

C

C Type declarations for local variables --------------------------------

C

INTEGER I, MEBAND, ML, MU

DOUBLE PRECISION DI, HRL1, ONE, PHRL1, R, ZERO

C-----------------------------------------------------------------------

C The following Fortran-77 declaration is to cause the values of the

C listed (local) variables to be saved between calls to this integrator.

C-----------------------------------------------------------------------

SAVE ONE, ZERO

C

COMMON /DVOD01/ ACNRM, CCMXJ, CONP, CRATE, DRC, EL(13),

1 ETA, ETAMAX, H, HMIN, HMXI, HNEW, HSCAL, PRL1,

2 RC, RL1, TAU(13), TQ(5), TN, UROUND,

3 ICF, INIT, IPUP, JCUR, JSTART, JSV, KFLAG, KUTH,

4 L, LMAX, LYH, LEWT, LACOR, LSAVF, LWM, LIWM,

5 LOCJS, MAXORD, METH, MITER, MSBJ, MXHNIL, MXSTEP,

6 N, NEWH, NEWQ, NHNIL, NQ, NQNYH, NQWAIT, NSLJ,

7 NSLP, NYH

C

DATA ONE /1.0D0/, ZERO /0.0D0/

C

IERSL = 0

GO TO (100, 100, 300, 400, 400), MITER

100 CALL DGESL (WM(3), N, N, IWM(31), X, 0)

RETURN

C

300 PHRL1 = WM(2)

HRL1 = H\*RL1

WM(2) = HRL1

IF (HRL1 .EQ. PHRL1) GO TO 330

R = HRL1/PHRL1

DO 320 I = 1,N

DI = ONE - R\*(ONE - ONE/WM(I+2))

IF (ABS(DI) .EQ. ZERO) GO TO 390

320 WM(I+2) = ONE/DI

C

330 DO 340 I = 1,N

340 X(I) = WM(I+2)\*X(I)

RETURN

390 IERSL = 1

RETURN

C

400 ML = IWM(1)

MU = IWM(2)

MEBAND = 2\*ML + MU + 1

CALL DGBSL (WM(3), MEBAND, N, ML, MU, IWM(31), X, 0)

RETURN

C----------------------- End of Subroutine DVSOL -----------------------

END

\*DECK DVSRCO

SUBROUTINE DVSRCO (RSAV, ISAV, JOB)

DOUBLE PRECISION RSAV

INTEGER ISAV, JOB

DIMENSION RSAV(\*), ISAV(\*)

C-----------------------------------------------------------------------

C Call sequence input -- RSAV, ISAV, JOB

C Call sequence output -- RSAV, ISAV

C COMMON block variables accessed -- All of /DVOD01/ and /DVOD02/

C

C Subroutines/functions called by DVSRCO.. None

C-----------------------------------------------------------------------

C This routine saves or restores (depending on JOB) the contents of the

C COMMON blocks DVOD01 and DVOD02, which are used internally by DVODE.

C

C RSAV = real array of length 49 or more.

C ISAV = integer array of length 41 or more.

C JOB = flag indicating to save or restore the COMMON blocks..

C JOB = 1 if COMMON is to be saved (written to RSAV/ISAV).

C JOB = 2 if COMMON is to be restored (read from RSAV/ISAV).

C A call with JOB = 2 presumes a prior call with JOB = 1.

C-----------------------------------------------------------------------

DOUBLE PRECISION RVOD1, RVOD2

INTEGER IVOD1, IVOD2

INTEGER I, LENIV1, LENIV2, LENRV1, LENRV2

C-----------------------------------------------------------------------

C The following Fortran-77 declaration is to cause the values of the

C listed (local) variables to be saved between calls to this integrator.

C-----------------------------------------------------------------------

SAVE LENRV1, LENIV1, LENRV2, LENIV2

C

COMMON /DVOD01/ RVOD1(48), IVOD1(33)

COMMON /DVOD02/ RVOD2(1), IVOD2(8)

DATA LENRV1/48/, LENIV1/33/, LENRV2/1/, LENIV2/8/

C

IF (JOB .EQ. 2) GO TO 100

DO 10 I = 1,LENRV1

10 RSAV(I) = RVOD1(I)

DO 15 I = 1,LENRV2

15 RSAV(LENRV1+I) = RVOD2(I)

C

DO 20 I = 1,LENIV1

20 ISAV(I) = IVOD1(I)

DO 25 I = 1,LENIV2

25 ISAV(LENIV1+I) = IVOD2(I)

C

RETURN

C

100 CONTINUE

DO 110 I = 1,LENRV1

110 RVOD1(I) = RSAV(I)

DO 115 I = 1,LENRV2

115 RVOD2(I) = RSAV(LENRV1+I)

C

DO 120 I = 1,LENIV1

120 IVOD1(I) = ISAV(I)

DO 125 I = 1,LENIV2

125 IVOD2(I) = ISAV(LENIV1+I)

C

RETURN

C----------------------- End of Subroutine DVSRCO ----------------------

END

\*DECK DEWSET

SUBROUTINE DEWSET (N, ITOL, RTOL, ATOL, YCUR, EWT)

DOUBLE PRECISION RTOL, ATOL, YCUR, EWT

INTEGER N, ITOL

DIMENSION RTOL(\*), ATOL(\*), YCUR(N), EWT(N)

C-----------------------------------------------------------------------

C Call sequence input -- N, ITOL, RTOL, ATOL, YCUR

C Call sequence output -- EWT

C COMMON block variables accessed -- None

C

C Subroutines/functions called by DEWSET.. None

C-----------------------------------------------------------------------

C This subroutine sets the error weight vector EWT according to

C EWT(i) = RTOL(i)\*abs(YCUR(i)) + ATOL(i), i = 1,...,N,

C with the subscript on RTOL and/or ATOL possibly replaced by 1 above,

C depending on the value of ITOL.

C-----------------------------------------------------------------------

INTEGER I

C

GO TO (10, 20, 30, 40), ITOL

10 CONTINUE

DO 15 I = 1, N

15 EWT(I) = RTOL(1)\*ABS(YCUR(I)) + ATOL(1)

RETURN

20 CONTINUE

DO 25 I = 1, N

25 EWT(I) = RTOL(1)\*ABS(YCUR(I)) + ATOL(I)

RETURN

30 CONTINUE

DO 35 I = 1, N

35 EWT(I) = RTOL(I)\*ABS(YCUR(I)) + ATOL(1)

RETURN

40 CONTINUE

DO 45 I = 1, N

45 EWT(I) = RTOL(I)\*ABS(YCUR(I)) + ATOL(I)

RETURN

C----------------------- End of Subroutine DEWSET ----------------------

END

\*DECK DVNORM

DOUBLE PRECISION FUNCTION DVNORM (N, V, W)

DOUBLE PRECISION V, W

INTEGER N

DIMENSION V(N), W(N)

C-----------------------------------------------------------------------

C Call sequence input -- N, V, W

C Call sequence output -- None

C COMMON block variables accessed -- None

C

C Subroutines/functions called by DVNORM.. None

C-----------------------------------------------------------------------

C This function routine computes the weighted root-mean-square norm

C of the vector of length N contained in the array V, with weights

C contained in the array W of length N..

C DVNORM = sqrt( (1/N) \* sum( V(i)\*W(i) )\*\*2 )

C-----------------------------------------------------------------------

DOUBLE PRECISION SUM

INTEGER I

C

SUM = 0.0D0

DO 10 I = 1, N

10 SUM = SUM + (V(I)\*W(I))\*\*2

DVNORM = SQRT(SUM/REAL(N))

RETURN

C----------------------- End of Function DVNORM ------------------------

END

\*DECK D1MACH

DOUBLE PRECISION FUNCTION D1MACH (IDUM)

INTEGER IDUM

C-----------------------------------------------------------------------

C This routine computes the unit roundoff of the machine.

C This is defined as the smallest positive machine number

C u such that 1.0 + u .ne. 1.0

C

C Subroutines/functions called by D1MACH.. None

C-----------------------------------------------------------------------

DOUBLE PRECISION U, COMP

U = 1.0D0

10 U = U\*0.5D0

COMP = 1.0D0 + U

IF (COMP .NE. 1.0D0) GO TO 10

D1MACH = U\*2.0D0

RETURN

C----------------------- End of Function D1MACH ------------------------

END

\*DECK XERRWD

SUBROUTINE XERRWD (MSG, NMES, NERR, LEVEL, NI, I1, I2, NR, R1, R2)

DOUBLE PRECISION R1, R2

INTEGER NMES, NERR, LEVEL, NI, I1, I2, NR

CHARACTER\*1 MSG(NMES)

C-----------------------------------------------------------------------

C Subroutines XERRWD, XSETF, XSETUN, and the function routine IXSAV,

C as given here, constitute a simplified version of the SLATEC error

C handling package.

C Written by A. C. Hindmarsh and P. N. Brown at LLNL.

C Version of 18 November, 1992.

C This version is in double precision.

C

C All arguments are input arguments.

C

C MSG = The message (character array).

C NMES = The length of MSG (number of characters).

C NERR = The error number (not used).

C LEVEL = The error level..

C 0 or 1 means recoverable (control returns to caller).

C 2 means fatal (run is aborted--see note below).

C NI = Number of integers (0, 1, or 2) to be printed with message.

C I1,I2 = Integers to be printed, depending on NI.

C NR = Number of reals (0, 1, or 2) to be printed with message.

C R1,R2 = Reals to be printed, depending on NR.

C

C Note.. this routine is machine-dependent and specialized for use

C in limited context, in the following ways..

C 1. The argument MSG is assumed to be of type CHARACTER, and

C the message is printed with a format of (1X,80A1).

C 2. The message is assumed to take only one line.

C Multi-line messages are generated by repeated calls.

C 3. If LEVEL = 2, control passes to the statement STOP

C to abort the run. This statement may be machine-dependent.

C 4. R1 and R2 are assumed to be in double precision and are printed

C in D21.13 format.

C

C For a different default logical unit number, change the data

C statement in function routine IXSAV.

C For a different run-abort command, change the statement following

C statement 100 at the end.

C-----------------------------------------------------------------------

C Subroutines called by XERRWD.. None

C Function routine called by XERRWD.. IXSAV

C-----------------------------------------------------------------------

C

INTEGER I, LUNIT, IXSAV, MESFLG

C

C Get logical unit number and message print flag. ----------------------

LUNIT = IXSAV (1, 0, .FALSE.)

MESFLG = IXSAV (2, 0, .FALSE.)

IF (MESFLG .EQ. 0) GO TO 100

C Write the message. ---------------------------------------------------

WRITE (LUNIT,10) (MSG(I),I=1,NMES)

10 FORMAT(1X,80A1)

IF (NI .EQ. 1) WRITE (LUNIT, 20) I1

20 FORMAT(6X,'In above message, I1 =',I10)

IF (NI .EQ. 2) WRITE (LUNIT, 30) I1,I2

30 FORMAT(6X,'In above message, I1 =',I10,3X,'I2 =',I10)

IF (NR .EQ. 1) WRITE (LUNIT, 40) R1

40 FORMAT(6X,'In above message, R1 =',D21.13)

IF (NR .EQ. 2) WRITE (LUNIT, 50) R1,R2

50 FORMAT(6X,'In above, R1 =',D21.13,3X,'R2 =',D21.13)

C Abort the run if LEVEL = 2. ------------------------------------------

100 IF (LEVEL .NE. 2) RETURN

STOP

C----------------------- End of Subroutine XERRWD ----------------------

END

\*DECK XSETUN

SUBROUTINE XSETUN (LUN)

C-----------------------------------------------------------------------

C This routine resets the logical unit number for messages.

C

C Subroutines called by XSETUN.. None

C Function routine called by XSETUN.. IXSAV

C-----------------------------------------------------------------------

INTEGER LUN, JUNK, IXSAV

C

IF (LUN .GT. 0) JUNK = IXSAV (1,LUN,.TRUE.)

RETURN

C----------------------- End of Subroutine XSETUN ----------------------

END

\*DECK XSETF

SUBROUTINE XSETF (MFLAG)

C-----------------------------------------------------------------------

C This routine resets the print control flag MFLAG.

C

C Subroutines called by XSETF.. None

C Function routine called by XSETF.. IXSAV

C-----------------------------------------------------------------------

INTEGER MFLAG, JUNK, IXSAV

C

IF (MFLAG .EQ. 0 .OR. MFLAG .EQ. 1) JUNK = IXSAV (2,MFLAG,.TRUE.)

RETURN

C----------------------- End of Subroutine XSETF -----------------------

END

\*DECK IXSAV

INTEGER FUNCTION IXSAV (IPAR, IVALUE, ISET)

LOGICAL ISET

INTEGER IPAR, IVALUE

C-----------------------------------------------------------------------

C IXSAV saves and recalls one of two error message parameters:

C LUNIT, the logical unit number to which messages are printed, and

C MESFLG, the message print flag.

C This is a modification of the SLATEC library routine J4SAVE.

C

C Saved local variables..

C LUNIT = Logical unit number for messages.

C The default is 6 (machine-dependent).

C MESFLG = Print control flag..

C 1 means print all messages (the default).

C 0 means no printing.

C

C On input..

C IPAR = Parameter indicator (1 for LUNIT, 2 for MESFLG).

C IVALUE = The value to be set for the parameter, if ISET = .TRUE.

C ISET = Logical flag to indicate whether to read or write.

C If ISET = .TRUE., the parameter will be given

C the value IVALUE. If ISET = .FALSE., the parameter

C will be unchanged, and IVALUE is a dummy argument.

C

C On return..

C IXSAV = The (old) value of the parameter.

C

C Subroutines/functions called by IXSAV.. None

C-----------------------------------------------------------------------

INTEGER LUNIT, MESFLG

C-----------------------------------------------------------------------

C The following Fortran-77 declaration is to cause the values of the

C listed (local) variables to be saved between calls to this routine.

C-----------------------------------------------------------------------

SAVE LUNIT, MESFLG

DATA LUNIT/6/, MESFLG/1/

C

IF (IPAR .EQ. 1) THEN

IXSAV = LUNIT

IF (ISET) LUNIT = IVALUE

ENDIF

C

IF (IPAR .EQ. 2) THEN

IXSAV = MESFLG

IF (ISET) MESFLG = IVALUE

ENDIF

C

RETURN

C----------------------- End of Function IXSAV -------------------------

END

C=======================================================================

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C BEGIN NUMERICAL SUBROUTINES REQUIRED BY DVODE C

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

subroutine dcopy(n,dx,incx,dy,incy)

c

c copies a vector, x, to a vector, y.

c uses unrolled loops for increments equal to one.

c jack dongarra, linpack, 3/11/78.

c modified 12/3/93, array(1) declarations changed to array(\*)

c

double precision dx(\*),dy(\*)

integer i,incx,incy,ix,iy,m,mp1,n

c

if(n.le.0)return

if(incx.eq.1.and.incy.eq.1)go to 20

c

c code for unequal increments or equal increments

c not equal to 1

c

ix = 1

iy = 1

if(incx.lt.0)ix = (-n+1)\*incx + 1

if(incy.lt.0)iy = (-n+1)\*incy + 1

do 10 i = 1,n

dy(iy) = dx(ix)

ix = ix + incx

iy = iy + incy

10 continue

return

c

c code for both increments equal to 1

c

c

c clean-up loop

c

20 m = mod(n,7)

if( m .eq. 0 ) go to 40

do 30 i = 1,m

dy(i) = dx(i)

30 continue

if( n .lt. 7 ) return

40 mp1 = m + 1

do 50 i = mp1,n,7

dy(i) = dx(i)

dy(i + 1) = dx(i + 1)

dy(i + 2) = dx(i + 2)

dy(i + 3) = dx(i + 3)

dy(i + 4) = dx(i + 4)

dy(i + 5) = dx(i + 5)

dy(i + 6) = dx(i + 6)

50 continue

return

end

C

C---

C

subroutine daxpy(n,da,dx,incx,dy,incy)

c

c constant times a vector plus a vector.

c uses unrolled loops for increments equal to one.

c jack dongarra, linpack, 3/11/78.

c modified 12/3/93, array(1) declarations changed to array(\*)

c

double precision dx(\*),dy(\*),da

integer i,incx,incy,ix,iy,m,mp1,n

c

if(n.le.0)return

if (da .eq. 0.0d0) return

if(incx.eq.1.and.incy.eq.1)go to 20

c

c code for unequal increments or equal increments

c not equal to 1

c

ix = 1

iy = 1

if(incx.lt.0)ix = (-n+1)\*incx + 1

if(incy.lt.0)iy = (-n+1)\*incy + 1

do 10 i = 1,n

dy(iy) = dy(iy) + da\*dx(ix)

ix = ix + incx

iy = iy + incy

10 continue

return

c

c code for both increments equal to 1

c

c

c clean-up loop

c

20 m = mod(n,4)

if( m .eq. 0 ) go to 40

do 30 i = 1,m

dy(i) = dy(i) + da\*dx(i)

30 continue

if( n .lt. 4 ) return

40 mp1 = m + 1

do 50 i = mp1,n,4

dy(i) = dy(i) + da\*dx(i)

dy(i + 1) = dy(i + 1) + da\*dx(i + 1)

dy(i + 2) = dy(i + 2) + da\*dx(i + 2)

dy(i + 3) = dy(i + 3) + da\*dx(i + 3)

50 continue

return

end

C

C---

C

subroutine dscal(n,da,dx,incx)

c

c scales a vector by a constant.

c uses unrolled loops for increment equal to one.

c jack dongarra, linpack, 3/11/78.

c modified 3/93 to return if incx .le. 0.

c modified 12/3/93, array(1) declarations changed to array(\*)

c

double precision da,dx(\*)

integer i,incx,m,mp1,n,nincx

c

if( n.le.0 .or. incx.le.0 )return

if(incx.eq.1)go to 20

c

c code for increment not equal to 1

c

nincx = n\*incx

do 10 i = 1,nincx,incx

dx(i) = da\*dx(i)

10 continue

return

c

c code for increment equal to 1

c

c

c clean-up loop

c

20 m = mod(n,5)

if( m .eq. 0 ) go to 40

do 30 i = 1,m

dx(i) = da\*dx(i)

30 continue

if( n .lt. 5 ) return

40 mp1 = m + 1

do 50 i = mp1,n,5

dx(i) = da\*dx(i)

dx(i + 1) = da\*dx(i + 1)

dx(i + 2) = da\*dx(i + 2)

dx(i + 3) = da\*dx(i + 3)

dx(i + 4) = da\*dx(i + 4)

50 continue

return

end

C

C---

C

subroutine dgefa(a,lda,n,ipvt,info)

integer lda,n,ipvt(n),info

double precision a(lda,n)

c

c dgefa factors a double precision matrix by gaussian elimination.

c

c dgefa is usually called by dgeco, but it can be called

c directly with a saving in time if rcond is not needed.

c (time for dgeco) = (1 + 9/n)\*(time for dgefa) .

c

c on entry

c

c a double precision(lda, n)

c the matrix to be factored.

c

c lda integer

c the leading dimension of the array a .

c

c n integer

c the order of the matrix a .

c

c on return

c

c a an upper triangular matrix and the multipliers

c which were used to obtain it.

c the factorization can be written a = l\*u where

c l is a product of permutation and unit lower

c triangular matrices and u is upper triangular.

c

c ipvt integer(n)

c an integer vector of pivot indices.

c

c info integer

c = 0 normal value.

c = k if u(k,k) .eq. 0.0 . this is not an error

c condition for this subroutine, but it does

c indicate that dgesl or dgedi will divide by zero

c if called. use rcond in dgeco for a reliable

c indication of singularity.

c

c linpack. this version dated 08/14/78 .

c cleve moler, university of new mexico, argonne national lab.

c

c subroutines and functions

c

c blas daxpy,dscal,idamax

c

c internal variables

c

double precision t

integer idamax,j,k,kp1,l,nm1

c

c

c gaussian elimination with partial pivoting

c

info = 0

nm1 = n - 1

if (nm1 .lt. 1) go to 70

do 60 k = 1, nm1

kp1 = k + 1

c

c find l = pivot index

c

l = idamax(n-k+1,a(k,k),1) + k - 1

ipvt(k) = l

c

c zero pivot implies this column already triangularized

c

if (a(l,k) .eq. 0.0d0) go to 40

c

c interchange if necessary

c

if (l .eq. k) go to 10

t = a(l,k)

a(l,k) = a(k,k)

a(k,k) = t

10 continue

c

c compute multipliers

c

t = -1.0d0/a(k,k)

call dscal(n-k,t,a(k+1,k),1)

c

c row elimination with column indexing

c

do 30 j = kp1, n

t = a(l,j)

if (l .eq. k) go to 20

a(l,j) = a(k,j)

a(k,j) = t

20 continue

call daxpy(n-k,t,a(k+1,k),1,a(k+1,j),1)

30 continue

go to 50

40 continue

info = k

50 continue

60 continue

70 continue

ipvt(n) = n

if (a(n,n) .eq. 0.0d0) info = n

return

end

C

C---

C

subroutine dgbsl(abd,lda,n,ml,mu,ipvt,b,job)

integer lda,n,ml,mu,ipvt(1),job

double precision abd(lda,1),b(1)

c

c dgbsl solves the double precision band system

c a \* x = b or trans(a) \* x = b

c using the factors computed by dgbco or dgbfa.

c

c on entry

c

c abd double precision(lda, n)

c the output from dgbco or dgbfa.

c

c lda integer

c the leading dimension of the array abd .

c

c n integer

c the order of the original matrix.

c

c ml integer

c number of diagonals below the main diagonal.

c

c mu integer

c number of diagonals above the main diagonal.

c

c ipvt integer(n)

c the pivot vector from dgbco or dgbfa.

c

c b double precision(n)

c the right hand side vector.

c

c job integer

c = 0 to solve a\*x = b ,

c = nonzero to solve trans(a)\*x = b , where

c trans(a) is the transpose.

c

c on return

c

c b the solution vector x .

c

c error condition

c

c a division by zero will occur if the input factor contains a

c zero on the diagonal. technically this indicates singularity

c but it is often caused by improper arguments or improper

c setting of lda . it will not occur if the subroutines are

c called correctly and if dgbco has set rcond .gt. 0.0

c or dgbfa has set info .eq. 0 .

c

c to compute inverse(a) \* c where c is a matrix

c with p columns

c call dgbco(abd,lda,n,ml,mu,ipvt,rcond,z)

c if (rcond is too small) go to ...

c do 10 j = 1, p

c call dgbsl(abd,lda,n,ml,mu,ipvt,c(1,j),0)

c 10 continue

c

c linpack. this version dated 08/14/78 .

c cleve moler, university of new mexico, argonne national lab.

c

c subroutines and functions

c

c blas daxpy,ddot

c fortran min0

c

c internal variables

c

double precision ddot,t

integer k,kb,l,la,lb,lm,m,nm1

c

m = mu + ml + 1

nm1 = n - 1

if (job .ne. 0) go to 50

c

c job = 0 , solve a \* x = b

c first solve l\*y = b

c

if (ml .eq. 0) go to 30

if (nm1 .lt. 1) go to 30

do 20 k = 1, nm1

lm = min0(ml,n-k)

l = ipvt(k)

t = b(l)

if (l .eq. k) go to 10

b(l) = b(k)

b(k) = t

10 continue

call daxpy(lm,t,abd(m+1,k),1,b(k+1),1)

20 continue

30 continue

c

c now solve u\*x = y

c

do 40 kb = 1, n

k = n + 1 - kb

b(k) = b(k)/abd(m,k)

lm = min0(k,m) - 1

la = m - lm

lb = k - lm

t = -b(k)

call daxpy(lm,t,abd(la,k),1,b(lb),1)

40 continue

go to 100

50 continue

c

c job = nonzero, solve trans(a) \* x = b

c first solve trans(u)\*y = b

c

do 60 k = 1, n

lm = min0(k,m) - 1

la = m - lm

lb = k - lm

t = ddot(lm,abd(la,k),1,b(lb),1)

b(k) = (b(k) - t)/abd(m,k)

60 continue

c

c now solve trans(l)\*x = y

c

if (ml .eq. 0) go to 90

if (nm1 .lt. 1) go to 90

do 80 kb = 1, nm1

k = n - kb

lm = min0(ml,n-k)

b(k) = b(k) + ddot(lm,abd(m+1,k),1,b(k+1),1)

l = ipvt(k)

if (l .eq. k) go to 70

t = b(l)

b(l) = b(k)

b(k) = t

70 continue

80 continue

90 continue

100 continue

return

end

C

C---

C

subroutine dgesl(a,lda,n,ipvt,b,job)

integer lda,n,ipvt(n),job

double precision a(lda,n),b(n)

c

c dgesl solves the double precision system

c a \* x = b or trans(a) \* x = b

c using the factors computed by dgeco or dgefa.

c

c on entry

c

c a double precision(lda, n)

c the output from dgeco or dgefa.

c

c lda integer

c the leading dimension of the array a .

c

c n integer

c the order of the matrix a .

c

c ipvt integer(n)

c the pivot vector from dgeco or dgefa.

c

c b double precision(n)

c the right hand side vector.

c

c job integer

c = 0 to solve a\*x = b ,

c = nonzero to solve trans(a)\*x = b where

c trans(a) is the transpose.

c

c on return

c

c b the solution vector x .

c

c error condition

c

c a division by zero will occur if the input factor contains a

c zero on the diagonal. technically this indicates singularity

c but it is often caused by improper arguments or improper

c setting of lda . it will not occur if the subroutines are

c called correctly and if dgeco has set rcond .gt. 0.0

c or dgefa has set info .eq. 0 .

c

c to compute inverse(a) \* c where c is a matrix

c with p columns

c call dgeco(a,lda,n,ipvt,rcond,z)

c if (rcond is too small) go to ...

c do 10 j = 1, p

c call dgesl(a,lda,n,ipvt,c(1,j),0)

c 10 continue

c

c linpack. this version dated 08/14/78 .

c cleve moler, university of new mexico, argonne national lab.

c

c subroutines and functions

c

c blas daxpy,ddot

c

c internal variables

c

double precision ddot,t

integer k,kb,l,nm1

c

nm1 = n - 1

if (job .ne. 0) go to 50

c

c job = 0 , solve a \* x = b

c first solve l\*y = b

c

if (nm1 .lt. 1) go to 30

do 20 k = 1, nm1

l = ipvt(k)

t = b(l)

if (l .eq. k) go to 10

b(l) = b(k)

b(k) = t

10 continue

call daxpy(n-k,t,a(k+1,k),1,b(k+1),1)

20 continue

30 continue

c

c now solve u\*x = y

c

do 40 kb = 1, n

k = n + 1 - kb

b(k) = b(k)/a(k,k)

t = -b(k)

call daxpy(k-1,t,a(1,k),1,b(1),1)

40 continue

go to 100

50 continue

c

c job = nonzero, solve trans(a) \* x = b

c first solve trans(u)\*y = b

c

do 60 k = 1, n

t = ddot(k-1,a(1,k),1,b(1),1)

b(k) = (b(k) - t)/a(k,k)

60 continue

c

c now solve trans(l)\*x = y

c

if (nm1 .lt. 1) go to 90

do 80 kb = 1, nm1

k = n - kb

b(k) = b(k) + ddot(n-k,a(k+1,k),1,b(k+1),1)

l = ipvt(k)

if (l .eq. k) go to 70

t = b(l)

b(l) = b(k)

b(k) = t

70 continue

80 continue

90 continue

100 continue

return

end

C

C---

C

subroutine dgbfa(abd,lda,n,ml,mu,ipvt,info)

integer lda,n,ml,mu,ipvt(1),info

double precision abd(lda,1)

c

c dgbfa factors a double precision band matrix by elimination.

c

c dgbfa is usually called by dgbco, but it can be called

c directly with a saving in time if rcond is not needed.

c

c on entry

c

c abd double precision(lda, n)

c contains the matrix in band storage. the columns

c of the matrix are stored in the columns of abd and

c the diagonals of the matrix are stored in rows

c ml+1 through 2\*ml+mu+1 of abd .

c see the comments below for details.

c

c lda integer

c the leading dimension of the array abd .

c lda must be .ge. 2\*ml + mu + 1 .

c

c n integer

c the order of the original matrix.

c

c ml integer

c number of diagonals below the main diagonal.

c 0 .le. ml .lt. n .

c

c mu integer

c number of diagonals above the main diagonal.

c 0 .le. mu .lt. n .

c more efficient if ml .le. mu .

c on return

c

c abd an upper triangular matrix in band storage and

c the multipliers which were used to obtain it.

c the factorization can be written a = l\*u where

c l is a product of permutation and unit lower

c triangular matrices and u is upper triangular.

c

c ipvt integer(n)

c an integer vector of pivot indices.

c

c info integer

c = 0 normal value.

c = k if u(k,k) .eq. 0.0 . this is not an error

c condition for this subroutine, but it does

c indicate that dgbsl will divide by zero if

c called. use rcond in dgbco for a reliable

c indication of singularity.

c

c band storage

c

c if a is a band matrix, the following program segment

c will set up the input.

c

c ml = (band width below the diagonal)

c mu = (band width above the diagonal)

c m = ml + mu + 1

c do 20 j = 1, n

c i1 = max0(1, j-mu)

c i2 = min0(n, j+ml)

c do 10 i = i1, i2

c k = i - j + m

c abd(k,j) = a(i,j)

c 10 continue

c 20 continue

c

c this uses rows ml+1 through 2\*ml+mu+1 of abd .

c in addition, the first ml rows in abd are used for

c elements generated during the triangularization.

c the total number of rows needed in abd is 2\*ml+mu+1 .

c the ml+mu by ml+mu upper left triangle and the

c ml by ml lower right triangle are not referenced.

c

c linpack. this version dated 08/14/78 .

c cleve moler, university of new mexico, argonne national lab.

c

c subroutines and functions

c

c blas daxpy,dscal,idamax

c fortran max0,min0

c

c internal variables

c

double precision t

integer i,idamax,i0,j,ju,jz,j0,j1,k,kp1,l,lm,m,mm,nm1

c

c

m = ml + mu + 1

info = 0

c

c zero initial fill-in columns

c

j0 = mu + 2

j1 = min0(n,m) - 1

if (j1 .lt. j0) go to 30

do 20 jz = j0, j1

i0 = m + 1 - jz

do 10 i = i0, ml

abd(i,jz) = 0.0d0

10 continue

20 continue

30 continue

jz = j1

ju = 0

c

c gaussian elimination with partial pivoting

c

nm1 = n - 1

if (nm1 .lt. 1) go to 130

do 120 k = 1, nm1

kp1 = k + 1

c

c zero next fill-in column

c

jz = jz + 1

if (jz .gt. n) go to 50

if (ml .lt. 1) go to 50

do 40 i = 1, ml

abd(i,jz) = 0.0d0

40 continue

50 continue

c

c find l = pivot index

c

lm = min0(ml,n-k)

l = idamax(lm+1,abd(m,k),1) + m - 1

ipvt(k) = l + k - m

c

c zero pivot implies this column already triangularized

c

if (abd(l,k) .eq. 0.0d0) go to 100

c

c interchange if necessary

c

if (l .eq. m) go to 60

t = abd(l,k)

abd(l,k) = abd(m,k)

abd(m,k) = t

60 continue

c

c compute multipliers

c

t = -1.0d0/abd(m,k)

call dscal(lm,t,abd(m+1,k),1)

c

c row elimination with column indexing

c

ju = min0(max0(ju,mu+ipvt(k)),n)

mm = m

if (ju .lt. kp1) go to 90

do 80 j = kp1, ju

l = l - 1

mm = mm - 1

t = abd(l,j)

if (l .eq. mm) go to 70

abd(l,j) = abd(mm,j)

abd(mm,j) = t

70 continue

call daxpy(lm,t,abd(m+1,k),1,abd(mm+1,j),1)

80 continue

90 continue

go to 110

100 continue

info = k

110 continue

120 continue

130 continue

ipvt(n) = n

if (abd(m,n) .eq. 0.0d0) info = n

return

end

C

C---

C

integer function idamax(n,dx,incx)

c

c finds the index of element having max. absolute value.

c jack dongarra, linpack, 3/11/78.

c modified 3/93 to return if incx .le. 0.

c modified 12/3/93, array(1) declarations changed to array(\*)

c

double precision dx(\*),dmax

integer i,incx,ix,n

c

idamax = 0

if( n.lt.1 .or. incx.le.0 ) return

idamax = 1

if(n.eq.1)return

if(incx.eq.1)go to 20

c

c code for increment not equal to 1

c

ix = 1

dmax = dabs(dx(1))

ix = ix + incx

do 10 i = 2,n

if(dabs(dx(ix)).le.dmax) go to 5

idamax = i

dmax = dabs(dx(ix))

5 ix = ix + incx

10 continue

return

c

c code for increment equal to 1

c

20 dmax = dabs(dx(1))

do 30 i = 2,n

if(dabs(dx(i)).le.dmax) go to 30

idamax = i

dmax = dabs(dx(i))

30 continue

return

end

C

C---

C

double precision function ddot(n,dx,incx,dy,incy)

c

c forms the dot product of two vectors.

c uses unrolled loops for increments equal to one.

c jack dongarra, linpack, 3/11/78.

c modified 12/3/93, array(1) declarations changed to array(\*)

c

double precision dx(\*),dy(\*),dtemp

integer i,incx,incy,ix,iy,m,mp1,n

c

ddot = 0.0d0

dtemp = 0.0d0

if(n.le.0)return

if(incx.eq.1.and.incy.eq.1)go to 20

c

c code for unequal increments or equal increments

c not equal to 1

c

ix = 1

iy = 1

if(incx.lt.0)ix = (-n+1)\*incx + 1

if(incy.lt.0)iy = (-n+1)\*incy + 1

do 10 i = 1,n

dtemp = dtemp + dx(ix)\*dy(iy)

ix = ix + incx

iy = iy + incy

10 continue

ddot = dtemp

return

c

c code for both increments equal to 1

c

c

c clean-up loop

c

20 m = mod(n,5)

if( m .eq. 0 ) go to 40

do 30 i = 1,m

dtemp = dtemp + dx(i)\*dy(i)

30 continue

if( n .lt. 5 ) go to 60

40 mp1 = m + 1

do 50 i = mp1,n,5

dtemp = dtemp + dx(i)\*dy(i) + dx(i + 1)\*dy(i + 1) +

\* dx(i + 2)\*dy(i + 2) + dx(i + 3)\*dy(i + 3) + dx(i + 4)\*dy(i + 4)

50 continue

60 ddot = dtemp

return

end